Jack Dongarra
Kaj Madsen
Jerzy Wasniewski (Eds.)

LNCS 3732

Applied Parallel Computing

State of the Art in Scientific Computing

7th International Workshop, PARA 2004
Lyngby, Denmark, June 2004
Revised Selected Papers

Springer

Jack Dongarra Kaj Madsen
Jerzy Wasniewski (Eds.)

Applied Parallel Computing

State of the Art in Scientific Computing

7th International Workshop, PARA 2004
Lyngby, Denmark, June 20-23, 2004
Revised Selected Papers

Springer

Volume Editors

Jack Dongarra
University of Tennessee
Department of Computer Science
1122 Volunteer Blvd.
Knoxville, TN 37996-3450, USA
and
Oak Ridge National Laboratory
Computer Science and Mathematics Division
E-mail: dongarra@cs.utk.edu

Kaj Madsen
Jerzy Wasniewski
Technical University of Denmark
Informatics and Mathematical Modelling
Richard Petersens Plads, Building 321
2800 Kongens Lyngby, Denmark
E-mail: {km,jw}@imm.dtu.dk

Library of Congress Control Number: 2006920921

CR Subject Classification (1998): G.1-4, F.1-2, D.1-3, J.1

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN	0302-9743
ISBN-10	3-540-29067-2 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-29067-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Markus Richter, Heidelberg
Printed on acid-free paper SPIN: 11558958 06/3142 5 4 3 2 1 0

Preface

Introduction

The PARA workshops in the past were devoted to parallel computing methods in science and technology. There have been seven PARA meetings to date: PARA'94, PARA'95 and PARA'96 in Lyngby, Denmark, PARA'98 in Umeå, Sweden, PARA 2000 in Bergen, Norway, PARA 2002 in Espoo, Finland, and PARA 2004 again in Lyngby, Denmark. The first six meetings featured lectures in modern numerical algorithms, computer science, engineering, and industrial applications, all in the context of scientific parallel computing.

This meeting in the series, the PARA 2004 Workshop with the title "State of the Art in Scientific Computing", was held in Lyngby, Denmark, June 20–23, 2004. The PARA 2004 Workshop was organized by Jack Dongarra from the University of Tennessee and Oak Ridge National Laboratory, and Kaj Madsen and Jerzy Waśniewski from the Technical University of Denmark. The emphasis here was shifted to high-performance computing (HPC). The ongoing development of ever more advanced computers provides the potential for solving increasingly difficult computational problems. However, given the complexity of modern computer architectures, the task of realizing this potential needs careful attention. For example, the failure to exploit a computer's memory hierarchy can degrade performance badly. A main concern of HPC is the development of software that optimizes the performance of a given computer.

The high cost of state-of-the-art computers can be prohibitive for many workplaces, especially if there is only an occasional need for HPC. A solution to this problem can be network computing, where remote computing facilities are exploited via the Internet.

PARA 2004 featured invited talks, contributed talks, minisymposia, and software and hardware vendors. The first day, June 20, was devoted to two parallel tutorials. The minisymposia and contributed talks during the main part of the workshop, June 21–23, were scheduled in parallel sessions. All invited and contributed talks were noncommercial. The workshop attracted 230 speakers from all over the world.

The PARA 2006 Workshop with the title "State-of-the-Art in Scientific and Parallel Computing" will be held in Umeå (Sweden) on June 17–21, 2006.

Tutorials

Validated Scientific Computing Using Interval Analysis was organized by *George F. Corliss* from Marquette University (USA). This tutorial gave an introduction to concepts and patterns of interval analysis. It was assumed that the participants had had a first course in scientific computation, including floating-point arithmetic, error analysis, automatic differentiation, Gaussian elimination, Newton's method, numerical optimization, and Runge-Kutta methods for ODEs. The tutorial included lectures, with examples in MATLAB and Sun's Fortran 95, and a set of supervised, hands-on exercises.

Automatic Differentiation was organized by *Andrea Walther* from the Technical University of Dresden (Germany). This tutorial gave a detailed introduction to the chain rule based technique of automatic differentiation (AD) that provides first and higher derivatives without incurring truncation errors. Several examples illustrated the theoretical results. Some AD tools, selected as a reasonably representative sample, were tested in supervised, hands-on exercises.

Key Speakers

Richard P. Brent, Oxford University Computing Laboratory (UK), *Fast and Reliable Random Number Generators for Scientific Computing*. Fast and reliable pseudo-random number generators are required for simulation and other applications in scientific computing. Richard outlined the requirements for good uniform random number generators, and described a class of generators having very fast vector/parallel implementations with excellent statistical properties.

Bernd Dammann and **Henrik Madsen**, the Technical University of Denmark (Denmark), *High Performance Computing and the Importance of Code Tuning—Some Practical Experiences from Program Tuning at the DTU HPC Center*. This talk gave a short overview of the High Performance Computer installation at the Technical University of Denmark (DTU), as well as a summary of some code tuning experiments. It is easy to reduce the run time of an application for a given problem by buying a computer with a faster CPU (higher clock frequency). However, very often the same or even better speed-up of the code can be achieved by analyzing and tuning the code—without the need to invest in new hardware.

Jack Dongarra, the University of Tennessee and Oak Ridge National Laboratory (USA), *High Performance Computing Trends and Self Adapting Numerical Software (SANS)—Effort*. In this talk Jack looked at how high performance computing has changed over the last 10 years and predicted future trends. In addition, he advocated the need for self adapting software.

Iain Duff, the Rutherford Appleton Laboratory (UK) and CERFACS (France), *Partitioning and Parallelism in the Solution of Large Sparse Systems*. Iain first reviewed the various levels of parallelism that are available in the direct solution of large sparse linear systems. He also briefly considered iterative as well as direct methods in this study.

Fred Gustavson, the IBM T.J. Watson Research Center (USA), *Ideas for High Performance Linear Algebra Software*. In this talk Fred presented several ideas for the development of sequential and parallel HPC dense linear algebra software. The main results were obtained from the algorithms and architecture approach.

Per Christian Hansen, the Technical University of Denmark (Denmark), *Large-Scale Methods in Inverse Problems*. Inverse problems arise in geophysics, tomography, image deblurring and many other areas where the goal is to compute interior or hidden information from exterior data. This talk presented a survey of numerical methods and paradigms suited for large-scale inverse problems.

Bo Kågström, the University of Umeå (Sweden), *Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software*. Matrix computations are both fundamental and ubiquitous in computational science and its vast application areas. Along with the development of more advanced computer systems with complex memory hierarchies, there is a continuing demand for new algorithms and library software that efficiently utilize and adapt to new architecture features.

John K. Reid, the Rutherford Appleton Laboratory (UK), *Fortran Is Getting More and More Powerful*. There is much happening just now with respect to Fortran. The features of Fortran 2003 have been chosen and the standard is very near completion. John is the convener of the ISO Fortran Committee.

Peter Sloot, the University of Amsterdam (The Netherlands). *Scientific Computing in the Grid: A Biophysical Case Study*. Workers at the University of Amsterdam conducted computer simulation experiments in pre-operative planning of vascular reconstruction with a physician in the experimental loop. Peter showed new results from numerical simulations of blood flow with 3D cellular automata.

Zahari Zlatev, National Environmental Research Institute (Denmark), *Large-Scale Computations with the Unified Danish Eulerian Model*. The Unified Danish Eulerian Model (UNI-DEM) is a mathematical model for performing different comprehensive studies related to damaging effects from high pollution levels in Denmark and Europe. The model is described by a system of partial differential equations (PDEs).

Minisymposia

Interval Methods, organized by *Luke Achenie*, University of Connecticut (USA), *Vladik Kreinovich*, University of Texas at El Paso (USA), and *Kaj Madsen*, Technical University of Denmark (Denmark). In many practical problems there is a need to (a) solve systems of equations and inequalities, and/or (b) optimize some performance measure. The results obtained by conventional algorithms are either local or cannot be guaranteed. Interval analysis provides guaranteed approximations of the set of all the actual solutions of the problem. This ensures that no solution is missed. There were 21 speakers in this minisymposium.

Trends in Large Scale Computing, organized by *Scott B. Baden*, University of California at San Diego (USA). Software infrastructures for large scale computation often fail to realize the full potential afforded by technological advances, and the result is lost opportunities for making scientific discoveries. This minisymposium examined two important issues in software infrastructure for large scale computation: achieving scalability, and optimization through specialization. There were 5 speakers in this minisymposium.

High Performance Linear Algebra Algorithms, organized by *Fred G. Gustavson*, IBM T.J. Watson Research Center (USA), and *Jerzy Waśniewski*, Technical University of Denmark (Denmark). The algorithms of Linpack and Eispack and later LAPACK and ScaLAPACK have stood the test of time in terms of robustness and accuracy. The focus of this minisymposium was on explaining high performance versions of these algorithms. There were 7 speakers in this minisymposium.

Substructuring, Dimension Reduction and Applications, organized by *Zhaojun Bai*, University of California (USA) and *Rencang Li*, University of Kentucky USA. There are a variety of reasons to go for substructuring and dimension reduction in scientific computations and applications. Substructuring makes it possible to solve large and seemingly intractable computational problems by some kind of divide-and-conquer technique. It also offers a general methodology for parallelization. There were 12 speakers in this minisymposium.

Parallel Processing in Science and Engineering, organized by *Adam W. Bojańczyk*, Cornell University (USA). This minisymposium concerned selected aspects of parallel and distributing computing as they arise in engineering. Both non-traditional applications as well as relevant software tools were presented. There were 9 speakers in this minisymposium.

Distributed Computing: Tools, Paradigms and Infrastructures, organized by *Beniamino Di Martino*, *Rocco Aversa*, Second University of Naples (Italy), and *Laurence Tianruo Yang*, Francis Xavier University (Canada). The minisymposium presented recent advances in distributed computing technology, methodology and tools. The presentations featured a variety of topics ranging from mobile and location-aware computing to skeletons and high-level parallel languages, from programming environments and tools for Grid applications development and tuning, to distributed monitoring and security issues. There were 9 speakers in this minisymposium.

High-Performance Computing in Earth and Space Science, organized by *Peter Messmer*, Tech-X Corporation at Boulder (USA). High-performance computing facilities enable simulations of physical phenomena with ever increasing fidelity and accuracy. The range of resolved scales in a single simulation, as well as the number of physical processes included, yield results that can be directly compared with observational data. There were 7 speakers in this minisymposium.

Advanced Algorithms and Software Components for Scientific Computing, organized by *Padma Raghavan*, Pennsylvania State University (USA). This minisymposium concerned algorithms for sparse linear systems solution and function approximation and their implementation using advanced software architectures. Discussions emphasized the role of such techniques for improving the performance of long-running PDE-based simulations. There were 7 speakers in this minisymposium.

Software Engineering and Problem Solving Environments for Scientific Computing, organized by *José C. Cunha*, Universidade Nova de Lisboa (Portugal) and *Omer F. Rana*, Cardiff University (UK). The emergence of computational grids in the last few years provides new opportunities for the scientific community to undertake collaborative and multi-disciplinary research. The aim of this minisymposium was to bring together experts who have experience in developing software tools to support application scientists, and those who make use of these tools. There were 5 speakers in this minisymposium.

Runtime Software Techniques for Enabling High-Performance Applications, organized by *Masha Sosonkina*, Iowa State University (USA). Parallel computing platforms are advancing rapidly, both in speed and size. However, often only a fraction of the peak hardware performance is achieved by high-performance scientific applications. One way to cope with the changeability of hardware is to start creating applications able to adapt themselves "on-the-fly". The talks of the minisymposium discussed this issue from both the application-centric and system-centric viewpoints. There were 6 speakers in this minisymposium.

Sparse Direct Linear Solvers, organized by *Sivan Toledo*, Tel Aviv University (Israel). The matrices of most of the systems of linear algebraic equations arising from scientific and engineering applications are sparse. This minisymposium dealt with some modern algorithms for sparse direct linear solvers. There were 12 speakers in this minisymposium.

Treatment of Large Scientific Models, organized by *Krassimir Georgiev*, Bulgarian Academy of Science (Bulgaria) and *Zahari Zlatev*, National Environmental Research Institute (Denmark). The exploitation of new fast computers in the effort to avoid non-physical assumptions and, thus, to develop and run more reliable and robust large scientific models was the major topic of this minisymposium. There were 9 speakers in this minisymposium.

Performance Evaluation and Design of Hardware-Aware PDE Solvers, organized by *Markus Kowarschik* and *Frank Hülsemann*, University of Erlangen-Nuremberg (Germany). In an ideal situation, all performance optimization of computationally intensive software would take place automatically, allowing the researchers to concentrate on the development of more efficient methods rather than having to worry about performance. However, for the time being, the need to identify and remove the performance bottlenecks of computationally intensive codes remains. As an example of a class of computationally intensive problems, this minisymposium concentrated on the numerical solution of PDEs. There were 7 speakers in this minisymposium.

Computationally Expensive Methods in Statistics, organized by *Wolfgang Hartmann*, SAS Institute Inc. (USA) and *Paul Somerville*, University of Central Florida (USA). A two-dimensional data set with N observations (rows) and n variables (columns) and large scale data requires intensive computational work. Of course there may be even more dimensions of the data set. There were 5 speakers in this minisymposium.

Approaches or Methods of Security Engineering (AMSE), organized by *Taihoon Kim* and *Ho Yeol Kwon*, Kangwon National University (Korea). Security engineering software is needed for reducing security holes. The talks presented a number of methods for designing such software. There were 16 speakers in this minisymposium.

Contributed Talks

Some contributed talks were added to the minisymposium sessions. The others were organized in the following independent sessions: two sessions of "Grid and Network", two sessions of "HPC Applied to Security Problems", two sessions of "Clusters and Graphics", one session of "HPC Applied to Cryptology", one session of "ODEs, PDEs and Automatic Differentiation", one session of "Computer Tools", and a special session of "Computer Vendors".

Workshop Proceedings

The proceedings of the PARA 2004 Workshop are divided into two complementary publications, this Springer volume and the following report:

- J. Dongarra, K. Madsen and J. Waśniewski (Eds.)

- ▶ Complementary proceedings of the PARA 2004 Workshop on State-of-the-Art in Scientific Computing, Lyngby, Denmark, June, 2004.
- ▶ IMM-Technical report-2005-09.
- ▶ Informatics and Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, Denmark.
- ▶ URL: http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=3927

A list of those papers appearing in the report is given in this volume in Chapter II of the contributed talks.

Acknowledgments

The organizers are indebted to the PARA 2004 sponsors, whose support was vital to the success of the workshop. The main sponsor was the Department of Informatics and Mathematical Modelling of the Technical University of Denmark. The other sponsors were:

- HP High Performance Computing USA,
- NAG Numerical Algorithms Group Ltd. UK,
- Comsol A/S Denmark (MATLAB distributor),
- Sun Microsystems Denmark,
- UNI•C Danish Computing Center Denmark,
- Microsoft Denmark, and
- IBM (International Business Machines) Denmark.

The organizers would like to thank Kirsten Probst for her excellent work as PARA 2004 secretary. Dorthe Thøgersen, Henrik Krogh and other staff of the conference also provided very valuable help.

We are very grateful to Prof. Ho Yeol Kwon from the Kangwon National University, Electrical and Computer Engineering Department for taking many photos during the PARA 2004 conference. These are available at the PARA 2004 URL

http://www.imm.dtu.dk/~jw/para04/.

Thanks are also due to Vincent A. Barker for his kind assistance in the preparation of both the workshop and these proceedings.

The PARA 2004 conference ended on June 23, 2004. The evening of June 23, the eve of St. Hans Day, is celebrated in Denmark by the lighting of bonfires. We are indebted to Arriva Denmark A/S for making available two boats for the PARA 2004 participants, from which we could see some of the bonfires and admire beautiful Copenhagen.

Finally, we would like to thank the PARA 2004 referees for their careful evaluation of the workshop papers.

Jack Dongarra
Kaj Madsen
Jerzy Waśniewski

Table of Contents

Fast and Reliable Random Number Generators for Scientific Computing*

Richard P. Brent

Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD, UK
random@rpbrent.co.uk

Abstract. Fast and reliable pseudo-random number generators are required for simulation and other applications in Scientific Computing. We outline the requirements for good uniform random number generators, and describe a class of generators having very fast vector/parallel implementations with excellent statistical properties. We also discuss the problem of initialising random number generators, and consider how to combine two or more generators to give a better (though usually slower) generator.

1 Introduction

Monte Carlo methods are of great importance in simulation [36], computational finance, numerical integration, computational physics [13,24], etc. Due to Moore's Law and increases in parallelism, the statistical quality of random number generators is becoming even more important than in the past. A program running on a supercomputer might use 10^9 random numbers per second over a period of many hours (or even months in some cases), so 10^{16} or more random numbers might contribute to the result. Small correlations or other deficiencies in the random number generator could easily lead to spurious effects and invalidate the results of the computation, see e.g. [13,34].

Applications require random numbers with various distributions (e.g. normal, exponential, Poisson, . . .) but the algorithms used to generate these random numbers almost invariably require a good uniform random number generator. A notable exception is Wallace's method [7,39] for normally distributed numbers. In this paper we consider only the generation of uniformly distributed numbers. Usually we are concerned with *real* numbers u_n that are intended to be uniformly distributed on the interval $[0, 1)$. Sometimes it is convenient to consider *integers* U_n in some range $0 \leq U_n < m$. In this case we require $u_n = U_n/m$ to be (approximately) uniformly distributed.

Pseudo-random numbers generated in a deterministic fashion on a digital computer can not be truly random. What is required is that finite segments of the sequence $(u_0, u_1, \cdots)$ behave in a manner indistinguishable from a truly random sequence. In practice, this means that they pass all statistical tests that are relevant to the problem at hand. Since the problems to which a library routine will be applied are not known

* This work was supported in part by EPSRC grant GR/N35366.

in advance, random number generators in subroutine libraries should pass a number of stringent statistical tests (and not fail any) before being released for general use.

Random numbers generated by physical sources are available [1,2,15,23,37,38]. However, there are problems in generating such numbers sufficiently fast, and experience with them is insufficient to be confident of their statistical properties. Thus, for the present, we recommend treating such physical sources of random numbers with caution. They can be used to initialise (and perhaps periodically reinitialise) deterministic generators, and can be combined with deterministic generators by the algorithms considered in §6. In the following we restrict our attention to deterministic pseudo-random number generators.

A sequence $(u_0, u_1, \cdots)$ depending on a finite state must eventually be periodic, i.e. there is a positive integer ρ such that $u_{n+\rho} = u_n$ for all sufficiently large n. The minimal such ρ is called the *period*.

In §§2–3 we consider desiderata for random number generators. In §§4–5, we describe one popular class of random number generators. In §6 we discuss how to combine two or more generators to give a (hopefully) better generator. Finally, in §7 we briefly mention some implementations.

2 Requirements for Good Random Number Generators

Requirements for a good pseudo-random number generator have been discussed in many surveys, e.g. [5,9,11,17,20,22,25]. Due to space limitations we can not cover all aspects of random number generation here, but we shall attempt to summarize and comment briefly on the most important requirements. Of course, some of the requirements listed below may be irrelevant in certain applications. For example, there is often no need to skip ahead (§2.4). In some applications, such as Monte Carlo integration, it may be preferable to use numbers that definitely do not behave like random numbers: they are "quasi-random" rather than random [32].

2.1 Uniformity

The sequence of random numbers should pass statistical tests for uniformity of distribution. This is usually easy for deterministic generators implemented in software. For physical/hardware generators, the well-known technique of Von Neumann, or similar but more efficient techniques [16], can be used to extract uniform bits from a sequence of independent but possibly biased bits.

2.2 Independence

Subsequences of the full sequence $(u_0, u_1, \cdots)$ should be independent. Random numbers are often used to sample a d-dimensional space, so the sequence of d-tuples $(u_{dn}, u_{dn+1}, \ldots, u_{dn+d-1})$ should be uniformly distributed in the d-dimensional cube $[0, 1]^d$ for all "small" values of d (certainly for all $d \le 6$). For random number generators on parallel machines, the sequences generated on each processor should be independent.

2.3 Long Period

As mentioned above, a simulation might use 10^{16} random numbers. In such a case the period ρ must exceed 10^{16}. For many generators there are strong correlations between $u_0, u_1, \cdots$ and $u_m, u_{m+1}, \cdots$, where $m \approx \rho/2$ (and similarly for other simple fractions of the period). Thus, in practice the period should be *much* larger than the number of random numbers that will ever be used. A good rule of thumb is to use at most $\rho^{1/2}$ numbers. In fact, there are reasons, related to the *birthday spacings test* [28], for using at most $\rho^{1/3}$ numbers: see [22, §6].

2.4 Ability to Skip Ahead

If a simulation is to be run on a machine with several processors, or if a large simulation is to be performed on several independent machines, it is essential to ensure that the sequences of random numbers used by each processor are disjoint. Two methods of subdivision are commonly used. Suppose, for example, that we require 4 disjoint subsequences for a machine with 4 processors. One processor could use the subsequence $(u_0, u_4, u_8, \cdots)$, another the subsequence $(u_1, u_5, u_9, \cdots)$, etc. For efficiency each processor should be able to "skip over" the terms that it does not require.

Alternatively, processor j could use the subsequence $(u_{m_j}, u_{m_j+1}, \cdots)$, where the indices m_0, m_1, m_2, m_3 are sufficiently widely separated that the (finite) subsequences do not overlap, but this requires some efficient method of generating u_m for large m without generating all the intermediate values $u_1, \ldots, u_{m-1}$.

For generators satisfying a linear recurrence, it is possible to skip ahead by forming high powers of the appropriate matrix (see [22, §3.5] for details). However, it is not so well known that more efficient methods exist using generating functions. Essentially, we can replace matrix multiplications by polynomial multiplications. Multiplying two $r \times r$ matrices is much more expensive than multiplying two polynomials modulo a polynomial of degree r. Details are given in [4] and an implementation that is practical for r of the order of 10^6 is available [3].

2.5 Proper Initialization

The initialization of random number generators, especially those with a large amount of state information, is an important and often neglected topic. In some applications only a short sequence of random numbers is used after each initialization of the generator, so it is important that short sequences produced with different seeds are uncorrelated.

For example, suppose that a random number generator with seed s produces a sequence $(u_1^{(s)}, u_2^{(s)}, u_3^{(s)}, \ldots)$. If we use m different seeds $s_1, s_2, \ldots, s_m$ and generate n numbers from each seed, we get an $m \times n$ array U with elements $U_{i,j} = u_j^{(s_i)}$. We do not insist that the seeds are random – they could for example be consecutive integers.

Packages such as Marsaglia's *Diehard* [26] typically test a 1-D array of random numbers. We can generate a 1-D array by concatenating the rows (or columns) of U. Irrespective of how this is done, we would hope that the random numbers would pass the standard statistical tests. However, many current generators fail because they were intended for the case $m = 1$ (or small) and n large [14,18]. The other extreme is m large and $n = 1$. In this case we expect $u_1^{(s)}$ to behave like a pseudo-random *function* of s.

2.6 Unpredictability

In cryptographic applications, it is not sufficient for the sequence to pass standard statistical tests for randomness; it also needs to be *unpredictable* in the sense that there is no efficient deterministic algorithm for predicting u_n (with probability of success significantly greater than expected by chance) from $(u_0, u_1, \ldots, u_{n-1})$, unless n is so large that the prediction is infeasible.

At first sight it appears that unpredictability is not required in scientific applications. However, if a random number generator is predictable then we can always devise a statistical test (albeit an artificial one) that the generator will fail. Thus, it seems a wise precaution to use an unpredictable generator if the cost of doing so is not too high. We discuss techniques for this in §6.

Strictly speaking, unpredictability implies uniformity, independence, and a (very) long period. However, it seems worthwhile to state these simpler requirements separately.

2.7 Efficiency

It should be possible to implement the method efficiently so that only a few arithmetic operations are required to generate each random number, all vector/parallel capabilities of the machine are used, and overheads such as those for subroutine calls are minimal. Of course, efficiency tends to conflict with other requirements such as unpredictability, so a tradeoff is often involved.

2.8 Repeatability

For testing and development it is useful to be able to repeat a run with *exactly* the same sequence of random numbers as was used in an earlier run. This is usually easy if the sequence is restarted from the beginning (u_0). It may not be so easy if the sequence is to be restarted from some other value, say u_m for a large integer m, because this requires saving the state information associated with the random number generator.

2.9 Portability

Again, for testing and development purposes, it is useful to be able to generate *exactly* the same sequence of random numbers on two different machines, possibly with different wordlengths. This was more difficult to achieve in the past than it is nowadays, when nearly all computers have wordlengths of 32 or 64 bits, and their floating-point arithmetic satisfies the IEEE 754 standard.

3 Equidistribution

We should comment on the concept of *equidistribution*, which we have not listed as one of our requirements. Definitions and examples can be found in [22, §4.2] and in [30, §1.2].

Consider concatenating the leading v bits from k consecutive random numbers. According to [30], a random number generator is said to be k-distributed to v-bit accuracy if each of the 2^{kv} possible combinations of bits occurs the same number of times over a full period of the random number generator, except that the sequence of all zero bits is allowed to occur once less often. Some generators with period $\rho = 2^r$ or $2^r - 1$ can be proved to satisfy this condition for $kv \leq r$. This is fine in applications such as Monte Carlo integration. However, the probability that a periodic but otherwise random sequence will satisfy the condition is vanishingly small. If we perform a "chi-squared" test on the output of a k-distributed generator, the test will be failed because the value of χ^2 is too *small* !

To give a simply analogy: if I toss a fair coin 100 times, I expect to get about 50 heads and 50 tails, but I would be mildly surprised to get exactly the same number of heads as tails (the probability of this occurring is about 0.08). If (with the aid of a computer) I toss a fair coin 10^{12} times, I should be *very* surprised to get exactly the same number of heads as tails. (For $2n$ tosses, the probability of an equal number of heads and tails occurring is about $1/\sqrt{n\pi}$.) This is another reason for using at most $\sqrt{\rho}$ numbers from the full period of length ρ (compare §2.3).

4 Generalized Fibonacci Generators

In this section we describe a popular class of random number generators. For various generalizations, see [22].

Given a circular buffer of length r words (or bits), we can generate pseudo-random numbers from a linear or nonlinear recurrence

$$u_n = f(u_{n-1}, u_{n-2}, \ldots, u_{n-r}) \ .$$

For speed it is desirable that $f(u_{n-1}, u_{n-2}, \ldots, u_{n-r})$ depends explicitly on only a small number of its r arguments. An important case is the class of "generalized Fibonacci" or "lagged Fibonacci" random number generators [17].

Marsaglia [25] considers generators $F(r, s, \theta)$ that satisfy

$$U_n = U_{n-r} \, \theta \, U_{n-s} \mod m$$

for fixed "lags" r and s ($r > s > 0$) and $n \geq r$. Here m is a modulus (typically 2^w if w is the wordlength in bits), and θ is some binary operator, e.g. addition, subtraction, multiplication or "exclusive or". We abbreviate these operators by $+$, $-$, $*$ and $\oplus$ respectively. Generators using $\oplus$ are also called "linear feedback shift register" (LFSR) generators or "Tausworthe" generators. Usually U_n is normalised to give a floating-point number $u_n = U_n/m \in [0, 1)$.

It is possible to choose lags r, s so that the period ρ of the generalized Fibonacci generators $F(r, s, +)$ is a large prime p or a small multiple of such a prime. Typically, the period of the least-significant bit is p; because carries propagate from the least-significant bit into higher-order bits, the overall period is usually $2^{w-1}\rho$ for wordlength w. For example, [9, Table 1] gives several pairs (r, s) with $r > 10^6$. (The notation in [9] is different: $r + \delta$ corresponds to our r.)

There are several ways to improve the performance of generalized Fibonacci generators on statistical tests such as the Birthday Spacings and Generalized Triple tests [25,28].

The simplest is to include small odd integer multipliers α and β in the generalized Fibonacci recurrence, e.g.

$$U_n = \alpha U_{n-r} + \beta U_{n-s} \mod m \ .$$

Other ways to improve statistical properties (at the expense of speed) are to include more terms in the linear recurrence [19], to discard some members of the sequence [24], or to combine two or three generators in various ways (see §6).

With suitable choice of lags (r, s), the generalised Fibonacci generators satisfy the requirements of uniformity, long period, efficiency, and ability to skip ahead. Because there are only three terms in the recurrence, each number depends on only two previous numbers, so there may be difficulty satisfying the requirement for independence, at least if r is small. Because they are based on a linear recurrence, they do *not* satisfy the requirement for unpredictability. In §6 we show how to overcome these difficulties.

5 Short-Term and Long-Term Properties

When considering pseudo-random number generators, it is useful to consider their short-term and long-term properties separately.

short-term means properties that can be tested by inspection of relatively short segments of the full cycle. For example, suppose that a uniform random number generator is used to simulate throws of a dice. If consective sixes never occur in the output (or occur with probability much lower than expected), then the generator is faulty, and this can be tested by inspection of the results of a few hundred simulated throws. For a more subtle example, consider a single-bit LFSR generator of the form discussed in §4, with largest lag r, and period $2^r - 1$. We can form an $r \times r$ matrix from r^2 consecutive bits of output. This matrix is nonsingular (considered as a matrix over $\mathrm{GF}(2)$). However, the probability that a random $r \times r$ matrix over $\mathrm{GF}(2)$ is nonsingular is strictly less than 1 (about 0.289 for large r, see [8]). Thus, by inspecting $O(r^2)$ consecutive bits of the output, we can detect a short-term nonrandomness.

long-term means properties that can be tested by inspection of a full cycle (or a significant fraction of a full cycle). For example, a uniform random number generator might have a small bias, so the expected output is $1/2 + \varepsilon$ instead of $1/2$. This could be detected by taking a sample of size slightly larger than $1/\varepsilon^2$.

Generalized Fibonacci generators based on primitive trinomials generally have good long-term properties, but bad short-term properties. To improve the short-term properties we can use *tempering* (transforming the output vectors by a carefully-chosen linear transformation), as suggested by Matsumoto and Kurita (see [30] and [22, §4.5]), or the other devices mentioned in §4.

6 Improving Generators

In this section we consider how generators that suffer some defects can be improved.

6.1 Improving a Generator by "Decimation"

If $(x_0, x_1, \ldots)$ is generated by a 3-term recurrence, we can obtain a (hopefully better) sequence $(y_0, y_1, \ldots)$ by defining $y_j = x_{jp}$, where $p > 1$ is a suitable constant. In other words, use every p-th number and discard the others.

Consider the case $F(r, s, \oplus)$ with $w = 1$ (LFSR) and $p = 3$. (If $p = 2$, the y_j satisfy the same 3-term recurrence as the x_j.)

Using generating functions, it is easy to show that the y_j satisfy a 5-term recurrence. For example, if $x_n = x_{n-1} \oplus x_{n-127}$, then $y_n = y_{n-1} \oplus y_{n-43} \oplus y_{n-85} \oplus y_{n-127}$. A more elementary approach for $p \le 7$ is given in [40].

A possible improvement over simple decimation is decimation by blocks [24].

6.2 Combining Generators by Addition or Xor

We can combine some number K of generalized Fibonacci generators by addition (mod 2^w). If each component generator is defined by a primitive trinomial $T_k(x) = x^{r_k} + x^{s_k} + 1$, with distinct prime degrees r_k, then the combined generator has period at least $2^{w-1} \prod_{k=1}^{K} (2^{r_k} - 1)$ and satisfies a 3^K-term linear recurrence.

Because the speed of the combined generator decreases like $1/K$, we would probably take $K \le 3$ in practice. The case $K = 2$ seems to be better (and more efficient) than "decimation" with $p = 3$.

Alternatively, we can combine K generalized Fibonacci generators by bitwise "exclusive or" operating on w-bit words. This has the advantage of mixing different algebraic operations (assuming that addition mod 2^w is used in the generalized Fibonacci recurrence). Note that the least-significant bits will be the same for both methods.

6.3 Combining by Shuffling

Suppose that we have two pseudo-random sequences $X = (x_0, x_1, \ldots)$ and $Y = (y_0, y_1, \ldots)$. We can use a buffer V of size B say, fill the buffer using the sequence X, then use the sequence Y to generate indices into the buffer. If the index is j then the random number generator returns $V[j]$ and replaces $V[j]$ by the next number in the X sequence [17, Algorithm M].

In other words, we use one generator to shuffle the output of another generator. This seems to be as good (and about as fast) as combining two generators by addition. B should not be too small.

6.4 Combining by Shrinking

Coppersmith *et al* [12] suggested using one sequence to "shrink" another sequence.

Suppose we have two pseudo-random sequences $(x_0, x_1, \ldots)$ and $(y_0, y_1, \ldots)$, where $y_i \in \mathrm{GF}(2)$. Suppose $y_i = 1$ for $i = i_0, i_1, \ldots$ Define a sequence $(z_0, z_1, \ldots)$ to be the subsequence $(x_{i_0}, x_{i_1}, \ldots)$ of $(x_0, x_1, \ldots)$. In other words, one sequence of bits (y_i) is used to decide whether to "accept" or "reject" elements of another sequence (x_i). This is sometimes called "irregular decimation" (compare §6.1).

Combining two sequences by shrinking is slower than combining the sequences by $+$ or $\oplus$, but is less amenable to analysis based on linear algebra or generating functions, so is preferable in applications where the sequence needs to be unpredictable. Note that it is dangerous to take the x_i to be larger than a single bit. For example, if we tried to speed up the combination process by taking x_i to be a whole word, then the cryptographic security could be compromised.

7 Implementations

Several good random number generators are available. Following is a small sample, not intended to be exhaustive: Matsumoto and Nishimura's *Mersenne twister*, based on a primitive trinomial of degree 19937 with tempering to improve short-term properties [30]; L'Ecuyer's *maximally equidistributed combined LFSR generators* [21]; and the author's *ranut* (Fortran) and *xorgens* (C) generators [3]. The *xorgens* generators are simple, fast, have passed all statistical tests applied so far, and are based on a generalization of a recent idea of Marsaglia [27]. They are related to LFSR generators [6], but do not use trinomials, and can be implemented faster than most other LFSR generators because the degree r can be chosen to be a multiple of 64.

To close with a word of warning: all pseudo-random number generators fail some statistical tests – this is inevitable, since they are generated deterministically. It is ultimately the user's responsibility to ensure that the pseudo-random numbers appear sufficiently random for the application at hand.

References

1. Anonymous, *Random number generation and testing*, NIST, December 2000. `http://csrc.nist.gov/rng/`.
2. Anonymous, *True randomness on request*, University of Geneva and id Quantique, May 2004, `http://www.randomnumber.info`.
3. R. P. Brent, *Some uniform and normal random number generators,* ranut version 1.03 (January 2002) and xorgens version 2.01 (August 2004). Available from `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/random.html`.
4. R. P. Brent, On the periods of generalized Fibonacci recurrences, *Math. Comp.* **63** (1994), 389–401.
5. R. P. Brent, Random number generation and simulation on vector and parallel computers, *LNCS* **1470**, Springer-Verlag, Berlin, 1998, 1–20.
6. R. P. Brent, Note on Marsaglia's xorshift random number generators, *J. of Statistical Software* **11**, 5 (2004), 1–4. `http://www.jstatsoft.org`.
7. R. P. Brent, Some comments on C. S. Wallace's random number generators, *Computer J.*, to appear. Preprint at `http://www.comlab.ox.ac.uk/oucl/work/richard.brent/pub/pub213.html`.
8. R. P. Brent and B. D. McKay, Determinants and ranks of random matrices over Z_m, *Discrete Mathematics* **66** (1987), 35–49. $\cdots$`/pub094.html`.
9. R. P. Brent and P. Zimmermann, Random number generators with period divisible by a Mersenne prime, *LNCS* **2667**, Springer-Verlag, Berlin, 2003, 1–10. Preprint at $\cdots$`/pub211.html`.

10. R. P. Brent and P. Zimmermann, Algorithms for finding almost irreducible and almost primitive trinomials, in *Primes and Misdemeanours: Lectures in Honour of the Sixtieth Birthday of Hugh Cowie Williams*, Fields Institute, Toronto, 2004, 91–102. Preprint at ···/pub212.html .
11. P. D. Coddington, Random number generators for parallel computers, *The NHSE Review* **2** (1996). http://nhse.cs.rice.edu/NHSEreview/RNG/PRNGreview.ps.
12. D. Coppersmith, H. Krawczyk and Y. Mansour, The shrinking generator, *Advances in Crpyptology – CRYPTO'93, LNCS* **773**, Springer-Verlag, Berlin, 1994, 22–39.
13. A. M. Ferrenberg, D. P. Landau and Y. J. Wong, Monte Carlo simulations: hidden errors from "good" random number generators, *Phys. Review Letters* **69** (1992), 3382–3384.
14. P. Gimeno, *Problem with* ran_array, personal communication, 10 Sept. 2001.
15. M. Jakobsson, E. Shriver, B. K. Hillyer and A. Juels, A practical secure physical random bit generator, *Proc. Fifth ACM Conference on Computer and Communications Security*, November 1998. http://www.bell-labs.com/user/shriver/random.html.
16. A. Juels, M. Jakobsson, E. Shriver and B. K. Hillyer, How to turn loaded dice into fair coins, *IEEE Trans. on Information Theory* **46**, 2000, 911–921.
17. D. E. Knuth, *The Art of Computer Programming, Volume 2: Seminumerical Algorithms* (third edition), Addison-Wesley, Menlo Park, CA, 1998.
18. D. E. Knuth, *A better random number generator*, January 2002, http://www-cs-faculty.stanford.edu/~knuth/news02.html.
19. T. Kumada, H. Leeb, Y. Kurita and M. Matsumoto, New primitive t-nomials $(t = 3, 5)$ over $\mathrm{GF}(2)$ whose degree is a Mersenne exponent, *Math. Comp.* **69** (2000), 811–814. Corrigenda: *ibid* **71** (2002), 1337–1338.
20. P. L'Ecuyer, Random numbers for simulation, *Comm. ACM* **33**, 10 (1990), 85–97.
21. P. L'Ecuyer, Tables of maximally equidistributed combined LFSR generators, *Mathematics of Computation* **68** (1999), 261–269.
22. P. L'Ecuyer, Random number generation, Chapter 2 of *Handbook of Computational Statistics*, J. E. Gentle, W. Haerdle, and Y. Mori, eds., Springer-Verlag, 2004, 35–70.
23. W. Knight, Prize draw uses heat for random numbers, *New Scientist*, 17 August 2004. http://www.newscientist.com/news/news.jsp?id=ns99996289.
24. M. Lüscher, A portable high-quality random number generator for lattice field theory simulations, *Computer Physics Communications* **79** (1994), 100–110.
25. G. Marsaglia, A current view of random number generators, in *Computer Science and Statistics: The Interface*, Elsevier Science Publishers B. V.,1985, 3–10.
26. G. Marsaglia, *Diehard*, 1995. Available from http://stat.fsu.edu/~geo/.
27. G. Marsaglia, Xorshift RNGs, *J. of Statistical Software* **8**, 14 (2003), 1–9. http://www.jstatsoft.org.
28. G. Marsaglia and W. W. Tsang, Some difficult-to-pass tests of randomness *J. of Statistical Software* **7**, 3 (2002), 1–9. http://www.jstatsoft.org.
29. M. Mascagni, M. L. Robinson, D. V. Pryor and S. A. Cuccaro, Parallel pseudorandom number generation using additive lagged-Fibonacci recursions, *Lecture Notes in Statistics* **106**, Springer-Verlag, Berlin, 1995, 263–277.
30. M. Matsumoto and T. Nishimura, Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, *ACM Transactions on Modeling and Computer Simulations* **8**, 1998, 3–30. Also http://www.math.keio.ac.jp/~matumoto/emt.html.
31. A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, New York, 1997. http://cacr.math.uwaterloo.ca/hac/.
32. H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*, CBMS-NSF Regional Conference Series in Applied Mathematics **63**, SIAM, Philadelphia, 1992.

33. W. P. Petersen, Lagged Fibonacci series random number generators for the NEC SX-3, *Internat. J. High Speed Computing* **6** (1994), 387–398.
34. L. N. Shchur, J. R. Heringa and H. W. J. Blöte, Simulation of a directed random-walk model: the effect of pseudo-random-number correlations, *Physica A* **241** (1997), 579.
35. S. Tezuka, P. L'Ecuyer, and R. Couture, On the add-with-carry and subtract-with-borrow random number generators, *ACM Trans. on Modeling and Computer Simulation* **3** (1993), 315–331.
36. I. Vattulainen, T. Ala-Nissila and K. Kankaala, Physical tests for random numbers in simulations, *Phys. Review Letters* **73** (1994), 2513–2516.
37. J. Walker, *HotBits: Genuine random numbers, generated by radioactive decay*, Fourmilab Switzerland, April 2004. `http://www.fourmilab.ch/hotbits/`.
38. C. S. Wallace, Physically random generator, *Computer Systems Science and Engineering* **5** (1990), 82–88.
39. C. S. Wallace, Fast pseudo-random generators for normal and exponential variates, *ACM Trans. on Mathematical Software* **22** (1996), 119–127.
40. R. M. Ziff, Four-tap shift-register-sequence random-number generators, *Computers in Physics* **12** (1998), 385–392.

New Generalized Data Structures for Matrices Lead to a Variety of High Performance Dense Linear Algebra Algorithms

Fred G. Gustavson

IBM T.J. Watson Research Center, Yorktown Heights NY 10598, USA
fg2@us.ibm.com

Abstract. This paper is a condensation and continuation of [9]. We present a novel way to produce dense linear algebra factorization algorithms. The current state-of-the-art (SOA) dense linear algebra algorithms have a performance inefficiency and hence they give sub-optimal performance for most of Lapack's factorizations. We show that standard Fortran and C two dimensional arrays are the main reason for the inefficiency. For the other standard format (packed one dimensional arrays for symmetric and/or triangular matrices) the situation is much worse. We introduce RFP (Rectangular Full Packed) format which represent a packed array as a full array. This means that performance of Lapack's packed format routines becomes equal to or better than their full array counterparts. Returning to full format, we also show how to correct these performance inefficiencies by using new data structures (NDS) along with so-called kernel routines. The NDS generalize the current storage layouts for both standard layouts. We use the Algorithms and Architecture approach to justify why our new methods gives higher efficiency. The simplest forms of the new factorization algorithms are a direct generalization of the commonly used LINPACK algorithms. All programming for our NDS can be accomplished in standard Fortran, through the use of three- and four-dimensional arrays. Thus, no new compiler support is necessary. Combining RFP format with square blocking or just using SBP (Square Block Packed) format we are led to new high performance ways to produce ScaLapack type algorithms.

1 Introduction

The BLAS (Basic Linear Algebra Subroutines) were introduced to make the algorithms of dense linear algebra (DLA) performance-portable. There is a performance inefficiency of LAPACK algorithms and it suffices to discuss the level 3 BLAS, DGEMM (Double precision GEneral Matrix Matrix) to illustrate this fact.

In [2,7] design principles for producing a high performance "level 3" DGEMM BLAS are given. A key design principle for DGEMM is to partition its matrix operands into submatrices and then call an L1 kernel routine multiple times on its submatrix operands. The suffix i in L i stands for level i cache. L i is not to be confused with level i BLAS. Another key design principle is to change the data format of the submatrix operands so that each call to the L1 kernel can operate at or near the peak MFlops (Million FLoating point OPerations per Second) rate. This format change and subsequent change back to standard data format is a cause of a performance inefficiency in DGEMM. The DGEMM interface definition requires that its matrix operands be stored as standard Fortran or C two-dimensional arrays. Any Lapack factorization routine of a matrix, A, calls `DGEMM` multiple times with *all* its operands being submatrices of A. For each call data copy will be done; the principle inefficiency is therefore multiplied by this number of calls. However, this inefficiency can be ameliorated by using the NDS to create a substitute for `DGEMM`, e.g. its analogous L1 kernel routine, which does *not* require the aforementioned data copy.

We introduce NDS as a replacement for standard Fortran/C array storage. One of the key insights is to see that storing a matrix as a collection of submatrices (e.g., square blocks of size `NB`) leads to very high performance on today's, RISC type, processors. NDS order the blocks in standard Fortran/C order; i.e., store the blocks either in column-major or row-major order. However, see Section 2.1 ahead. The main benefit of the simpler data layout is that addressing of an arbitrary $a(i, j)$ element of matrix A can be easily handled by a compiler and/or a programmer. We call the NDS *simple* if the ordering of the blocks follows the standard row / column major order.

For level 3 algorithms, the basis of the ESSL (Engineering and Scientific Subroutine Library) are kernel routines that achieve peak performance when the underlying arrays fit into L1 cache [2]. If one were to adopt these new, simple NDS then BLAS and Lapack type algorithms become almost trivial to write. Also, the combination of using the NDS with kernel routines is a general procedure and for matrix factorization it helps to overcomes the current performance problems introduced by having a non-uniform, deep memory hierarchy. We use the AA (Algorithms and Architecture) approach, see [2], to illustrate what we mean. We shall make eight points below. Points 1 to 3 are commonly accepted architecture facts about many of today's processors. Points 4 to 6 are dense linear algebra algorithms facts that are easily demonstrated or proven. Points 7 and 8 are an obvious conclusion based on the AA approach.

1. Floating point arithmetic cannot be done unless the operands involved first reside in the L1 cache.
2. Two-dimensional Fortran and C arrays do *not* map nicely into L1 cache.
   (a) The best case happens when the array is contiguous and properly aligned.
   (b) At least a three-way set associative cache is required when matrix multiply is being done.
3. For peak performance, all matrix operands must be used multiple times when they enter L1 cache.
   (a) This assures that the cost of bringing an operand into cache is amortized by its level 3 multiple re-use.
   (b) Multiple re-use of all operands only occurs when all matrix operands map well into L1 cache.
4. Each scalar $a(i, j)$ factorization algorithm has a square submatrix counterpart
   `A(I:I+NB-1,J:J+NB-1)`
   algorithm.
   (a) Golub and Van Loan's "Matrix Computations" book.
   (b) The Lapack library.
5. Some submatrices are both contiguous and fit into L1 cache.
6. Dense matrix factorization is a level 3 computation.

   (a) Dense matrix factorization, in the context of point 4, is a series of submatrix computations.
   (b) Every submatrix computation (executing any kernel routine) is a level 3 computation that is done in the L1 cache.
   (c) A level 3 L1 computation is one in which each matrix operand gets used multiple times.
7. Map the input Fortran/C array (matrix A) to a set of contiguous submatrices each fitting into the L1 cache.
   (a) For portability (using block hybrid format (BHF)), perform the inverse map after applying point 8 (below).
8. Apply the appropriate submatrix algorithm.

The block submatrix codes of point 4b use Fortran and C to input their matrices, so point 5 does *not* hold for SOA algorithms. See page 739 of [8] for more details. Point 5 does hold for the NDS described here. Assuming both points 5 and 6 hold, we see that point 3 holds for every execution of the kernel routines that make up the factorization algorithm. This implies that near peak performance will be achieved. Point 7 is pure overhead for the new algorithms. Using the new data formats reduces this cost to zero. By only doing point 8 we see that we can get near peak performance as every subcomputation of point 8 is a point 6b computation.

Now we discuss the use of kernel routines in concert with NDS. Take any standard linear algebra factorization code, say Gaussian elimination with partial pivoting or the QR factorization of an `M` by `N` matrix, A. It is quite easy to derive the block equivalent code from the standard code. In the standard code a floating point operation is usually a Fused Multiply Add (FMA), ($c = c - ab$), whose block equivalent is a call to a `DGEMM` kernel. Similar analogies exist; e.g., for $b = b/a$ or $b = b * a$, we have a call to either a `DTRSM` or a `DTRMM` kernel. In the simple block equivalent codes we are led to one of the variants of IJK order. For these types of new algorithms the BLAS are simply calls to kernel routines. It is important to note that no data copying need be done.

There is one type of kernel routine that deserves special mention. It is the factor kernel. Neither Lapack nor the research literature treat factor kernels in sufficient depth. For example, the factor part of Lapack level 3 factor routines are level 2 routines; they are named with the suffix `TF2`, and they call level 2 BLAS repetitively. On the other hand, [2,3], and [8,5], where recursion is used, have produced level 3 factor routines that employ level 3 factor kernels to yield level 3 factor parts.

Besides full storage, there is packed storage, which is used to hold symmetric / triangular matrices. Using the NDS instead of the standard packed format [9,3] describes new algorithms that save "half" the storage of full format for symmetric matrices and outperform the current block based level 3 Lapack algorithms done on full format symmetric matrices. Also, we introduce RFP format which is a variant of hybrid full packed (HFP) format. HFP format is described in [6] of these proceedings. RFP format is a rearrangement of standard full storage holding a symmetric / triangular matrix A into a compact full storage rectangular array AR that uses minimal storage `NT=N(N+1)/2`. Therefore, level 3 BLAS can be used on AR. In fact, with the equivalent Lapack algorithm, using AR instead of A, gives slightly better performance. This offers the possibility to replace all packed or full Lapack routines with equivalent Lapack routines that work on AR. We present a new algorithm and indicate its performance for Cholesky factorization using AR instead of full A or packed AP.

Another generalization of using NDS applies or relates to ScaLapack. The standard block cyclic layout on a P by Q mesh of processors has parameter `NB`. Therefore, it is natural to view the square submatrices of order `NB` that arise in these layouts as atomic units. Now, many ScaLapack algorithms can be viewed as right looking Lapack algorithms: factor and scale a pivot panel, broadcast the scaled pivot panel to all processors, and then perform a Schur complement update on all processors. We describe in Section 4 some benefits: (1) since P and Q are arbitrary integers the square blocks can move about the mesh as contiguous atomic units; (2) it is possible to eliminate the PBLAS layer of ScaLapack as only standard Level 3 BLAS are needed; (3) for triangular / symmetric matrices one only needs to use about half the storage.

In Section 2 we describe some basic algorithmic and architectural results as a rationale for the work we are presenting. Section 2.1 describes a new concept which we call the L1 cache / L0 cache interface [4]. L0 cache is the register file of a Floating Point Unit. In Section 3 we describe SBP formats for symmetric/triangular arrays and show that they generalize both the standard packed and full arrays used by dense linear algebra algorithms. We also describe RFP format arrays. They can be used to replace both the standard packed and full arrays used by DLA algorithms. A minimal new coding effort is required as existing Lapack routines would constitute most of the new code.

2 Rationale and Underlying Foundations of Our Approach

For a set of linear equations $Ax = b$ there are two points of view. The more popular view is to select an algorithm, say Gaussian elimination with partial pivoting, and use it to compute x. The other view, which we adopt here, is to perform a series of linear transformations on both A and b so that the problem, in the new coordinate system, becomes simpler to solve. Both points of view have their merits. We use the second as it demonstrates some reasons why the AA approach, [2], is so effective. Briefly, the AA approach states that the key to performance is to understand the algorithm and architecture interaction. Furthermore a significant improvement in performance can be obtained by matching the algorithm to the architecture and vice-versa. In any case, it is a very cost-effective way of providing a given level of performance.

In [9] we show that performing two linear transformations R and S in succession defines matrix multiplication. Let $T = S(R)$ be linear and R and S have basis vectors. The basis of T, in terms of the bases of R and S defines matrix multiplication. Hence, taking view two above tells us that many DLAFA (Dense Linear Algebra Factorization Algorithms) are just performing matrix multiplication. Therefore, the AA approach directs our attention to optimize matrix multiplication and to isolate out all of its occurrences in each DLAFA. We now know that this is possible to do.

We end this section with brief remarks about blocking. The general idea of blocking is to get information to a high speed storage and use it multiple times to amortize the cost of moving the data. In doing so, we satisfy points 1 and 3 of the Introduction. We only touch upon TLB (Translation Look-aside Buffer), cache, and register blocking. The TLB contains a finite set of pages. These pages are known as the *current working set* of the computation. If the computation addresses only memory in the TLB then there is no penalty. Otherwise, a TLB miss occurs resulting in a large performance penalty; see [2]. Cache blocking reduces traffic between the memory and cache. Analogously, register blocking reduces traffic between cache and the registers of the CPU. Cache and register blocking are further discussed in [2].

2.1 The Need to Reorder a Contiguous Square Block

NDS are basically about using multiple SB (square block) of order `NB` to represent a matrix A. Each SB must be contiguous in memory. A contiguous block of memory maps best into L1 cache as it minimizes L1 and L2 cache misses as well as TLB misses for matrix multiply and other common row and column matrix operations; see [10]. By using SB's one can avoid the $O(N^3)$ amount of data copy in calling `DGEMM` repetitively in a DLAFA algorithm.

Recently computer manufacturers have introduced hardware that initiates multiple floating point operations (two to four) in a single cycle; eg [4]. However, each floating point operation requires several cycles (five to ten) to complete. Therefore, one needs to be able to schedule many (ten to forty) independent floating point operations every cycle if one wants to run their floating point applications at their peak (MFlops) rate. To make this possible hardware needs to introduce larger floating point register files (storage for the operands and results of the floating point units). We call this tiny memory the L0 cache. We now want to discuss a concept which we call the L1 cache / L0 cache interface. There are also new floating point multiple load and store instructions associated with the multiple floating point operations. A multiple load / store operation usually requires that its multiple operands be contiguous in L1 cache. Hence, data that enters L1 must also be properly ordered to be able to enter L0 in an optimal way. Unfortunately, layout of a SB in standard row / column major order *no longer* leads to an optimal way. Thus it is necessary to reorder a SB into submatrices which we call register blocks. Doing this produces a new data layout that will still be contiguous in L1 but also can be loaded into L0 from L1 in an optimal manner. The order and size in which the submatrices (register blocks) are chosen are platform dependent.

3 Square Blocked Packed and Rectangular Full Packed Formats for Symmetric/Triangular Arrays

SBP formats are a generalization of packed format for triangular arrays. They are also a generalization of full format for triangular arrays. The main benefit of the new formats is that they allow for level 3 performance while using about half the storage of the full array cases. In [6,9], there is a description of packed format layout.

For SBP formats there are two parameters, `NB` and `TRANS`, where, typically, $n \geq$ `NB`. For this format, we first choose a block size, `NB`, and then we lay out the data in squares of size `NB`. Each square block can be in row-major order (`TRANS` = 'T') or column-major order (`TRANS` = 'N'). This format supports both `uplo` = 'U' or 'L'. We only cover the case `uplo` = 'L'. For `uplo` = 'L', the first vertical stripe is n by `NB` and it consists of n_1 square blocks where $n_1 = \lceil n/\mathtt{NB} \rceil$. It holds the first trapezoidal n by `NB` part of L.

The next stripe has $n_1 - 1$ square blocks and it holds the next trapezoidal n - `NB` by `NB` part of L, and so on, until the last stripe consisting of the last leftover triangle is reached. There are $n_1(n_1 + 1)/2$ square blocks in all.

An example of Square Blocked Lower Packed Format (SBLP) with TRANS = 'T' is given in Figure 1 left. Here n = 10, `NB` = 4 and TRANS = 'T' and the numbers represent the position within the array where $a(i, j)$ is stored. Note the missing numbers (e.g., 2, 3, 4, 7, 8, and 12) which correspond to the upper right corner of the first stripe. This square blocked, lower, packed array consists of 6 square block arrays. The first three blocks hold submatrices that are 4 by 4, 4 by 4, and 2 by 4. The next two blocks hold submatrices that are 4 by 4 and 2 by 4. The last square block holds a 2 by 2 submatrix. Note the padding, which is done for ease of addressing. Addressing this set of six square blocks as a composite block array is straightforward.

```
 1  *  *  *
 5  6  *  *
 9 10 11  *
13 14 15 16
-----------|
17 18 19 20|49  *  *  *
21 22 23 24|53 54  *  *
25 26 27 28|57 58 59  *
29 30 31 32|61 62 63 64
-----------|-----------|
33 34 35 36|65 66 67 68|81  * * *
37 38 39 40|69 70 71 72|85 86 * *
 *  *  *  *| *  *  *  *| *  * * *
 *  *  *  *| *  *  *  *| *  * * *
```

```
 1  *  *  *  *  *  *  *   *   *
 2 14  *  *  *  *  *  *   *   *
 3 15 27  *  *  *  *  *   *   *
 4 16 28 40  *  *  *  *   *   *
 5 17 29 41 53  *  *  *   *   *
 6 18 30 42 54 66  *  *   *   *
 7 19 31 43 55 67 79  *   *   *
 8 20 32 44 56 68 80 92   *   *
 9 21 33 45 57 69 81 93 105   *
10 22 34 46 58 70 82 94 106 118
 *  *  *  *  *  *  *  *   *   *
 *  *  *  *  *  *  *  *   *   *
```

Fig. 1. Square Blocked Lower Packed Format for `NB=4` and `LDA=NB`

Now we turn to full format storage. We continue the example with `N` = 10, and `LDA` = 12. Simply set `NB` = `LDA` = 12 and one obtains full format; i.e., square block packed format gives a single block triangle which happens to be full format (see Figure 1 right). It should be clear that SBP format generalizes standard full format.

I believe a main innovation in using the SBP formats is to see that one can translate, verbatim, standard packed or full factorization algorithms into a SBP format algorithm by replacing each reference to an i, j element by a reference to its corresponding SB submatrix. This is an application of point 4 in the Introduction. Because of this storage layout, the beginning of each SB is easily located. Also key is that this format supports level 3 BLAS. Hence, old, packed and full codes are easily converted into square blocked, packed, level 3 code. In a nutshell, I am keeping "standard packed or full" addressing so the library writer/user can handle his own addressing in a Fortran/C environment. Performance results of SBP (and its variant Block Hybrid Format (BHF)) formats, for Cholesky factorization, are given [9,3].

3.1 RFP Format

RFP format is a standard full array of size `NT` that holds a symmetric / triangular matrix A. It is closely related to HFP (see [6] of this proceedings) format which represents A as the concatenation of two standard full arrays whose total size is also NT. The basic idea behind both formats is quite simple. Given an order n symmetric matrix $A = LL^T$, we break it into a block 2×2 form

$$A = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = LL^T = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix} \tag{1}$$

where A_{11} and A_{22} are symmetric and L_{11} and L_{22} are lower triangular. We need only store the lower triangles of A_{11} and A_{22} as well as the full matrix A_{21}. When $n = 2k$ is even, the lower triangle of A_{11} and the upper triangle of A_{22}^T can be concatenated together along their main diagonals into an $(k+1) \times k$ dense matrix. The offdiagonal block A_{21} is $k \times k$, and so it can be appended below the $(k+1) \times k$ dense matrix. Thus, the lower triangle of A can be stored as a single $(n+1) \times k$ dense matrix AR. In effect, each block matrix A_{11}, A_{21} and A_{22} is now stored in "full format", meaning its entries can be accessed with constant row and column strides. The full power of Lapack's block level 3 BLAS are now available for symmetric and triangular computations while still using the minimal amount of storage. Note that AR^T which is $k \times (n+1)$ also has these two desirable properties; see Figure 2 where $n = 10$. The performance of the *simple related partition algorithm* (SRPA) on RFP format is very similar to the SRPA on HFP format; see Figure 1 and Table 2 of [6] as space does not allow us to produce these results here.

```
        LRFP AR                        LRFP AR transpose

    55 65 75 85 95                55|00 10 20 30 40|50 60 70 80 90
    00|66 76 86 96                65 66|11 21 31 41|51 61 71 81 91
    10 11|77 87 97                75 76 77|22 32 42|52 62 72 82 92
    20 21 22|88 98                85 86 87 88|33 43|53 63 73 83 93
    30 31 32 33|99                95 96 97 98 99|44|54 64 74 84 94
    40 41 42 43 44
    50 51 52 53 54
    60 61 62 63 64
    70 71 72 73 74
    80 81 82 83 84
    90 91 92 93 94
```

Fig. 2. Lower Rectangular Full Packed formats when $n = 10$, `LDAR = n+1`

We now illustrate how to program Lapack algorithms for RFP format using existing Lapack routines and level 3 BLAS. The above SRPA with partition sizes k and k and $n = 2k$ is: (see equation 1 and Figure 2).

1. `call dpotrf('L',k,AR(1,0),n+1,info)` ! factor $L_{11}L_{11}^T = A_{11}$
2. ! solve $L_{21}L_{11}^T = A_{21}$
   `call dtrsm('R','L','T','N',k,k,one,AR(1,0),n+1,AR(k+1,0),n+1)`
3. ! update $A_{22} \leftarrow A_{22} - L_{21}L_{11}^T$
   `call dsyrk('U','T',k,k,-one,AR(0,0),n+1,one,AR(k+1,0),n+1)`
4. `call dpotrf('U',k,AR(0,0),n+1,info)` ! factor $L_{22}L_{22}^T = A_{22}$

This covers RFP format when `uplo` = ’L’ and n is even. A similar result holds for n odd. Also, for `uplo` = ’U’ and n both even and odd similar results hold. Because of space limitations we do not describe these cases.

4 Distributed Memory Computing (DMC) with SB and SBP Formats

We are concerned with a $P \times Q$ mesh of processors and the programming model is a block cyclic layout of a global rectangular array. Thus, symmetric and triangular arrays waste half the storage on all processors. Using the results of Section 3 we can save this wasted storage if we are willing to use SB of order NB^2. This is a natural thing to do as NB is the free parameter of our programming model. Let $n_1 = \lceil N/NB \rceil$ where N is the order of our global symmetric matrix A. From now on shall be concerned with A being a matrix of SB’s of block order n_1. Now P and Q *can* have nothing to do with n_1. This fact tells us that we should treat each SB of A as an *atomic unit* because when P and Q are relatively prime the SB’s of A will move about the processors as single contiguous blocks. We want these SB’s that move to be part of our data layout so that the Send / Receive buffers of MPI or the BLACs that ScaLapack uses can be treated as contiguous blocks of storage. This allows us to avoid copying matrix data to a Send buffer and copying a Receive buffer to matrix data.

We shall use a right looking algorithm (RLA) that is especially tailored to DMC. We explain it for the global matrix A. $A = (F_1U_1)(F_2U_2)\dots(F_{n_1}U_{n_1})$ where $U_{n_1} = I$ is one way to factor A. Here F_i, U_i are the factor and update parts at stage i of the RLA. Another way has $A = (F_1)(U_1F_2)\dots(U_{n_1-1}F_{n_1})$. This second way allows us to overlap F_{i+1} with U_i; see [1]. Let processor column (pc) J hold the pivot panel (pp) of F_{i+1}. Now $pc(J)$ will update $pp(F_{i+1})$ with U_i, factor $pp(F_{i+1})$, Send (Broadcast) $pp(F_{i+1:n_1})$ to all other $pc(K)$, $0 \le K < Q$ and finally will update its remaining column panels. Simultaneously, the remaining $pc(K), K \ne J$ will just update all of their column panels.

There are three separate sub algorithms: factor F_{i+1}, Send / Receive $F_{i+1:n_1}$ and update on $p(I,J)$ for all I and J. The update algorithm is called the Schur Complement Update: Each active SB on $p(I,J)$ gets a `DGEMM` update. What is missing are the `A,B` operands of `DGEMM`. We add these to our data structure by placing on each $p(I,J)$ West and South border vectors that will hold all of $p(I,J)$ SB `A,B` operands. These borders, now part of our data layout, are the Send / Receive buffers referred to above. Now `SB(il,jl)=SB(il,jl)-W(il)*S(jl)` becomes the generic `DGEMM` update on $p(I,J)$ where `il,jl` are local coordinates on $p(I,J)$. Thus, the Schur Complement update is just a sum of `DGEMM′s` over all active SB’s and our DMC paradigm guarantees almost perfect load balance.

There are two algorithms that emerge for A. One is based on the block version of RFP format. Since AR is rectangular, it should not be hard to see that each $p(I,J)$ holds a rectangular matrix. It is made up of pieces of two triangles `T1, T2` and a square `S1`; see [6] for the meaning of `T1, S1, T2`. And because the SB’s of `T2` are reflected in `T2`’s main diagonal, we also need to introduce North and East border vectors to hold the `A,B` operands of `T2` SB’s. The coding becomes intricate because AR consists of three distinct pieces of A. The second algorithm is simpler to code. Global A consists of `NT1` = $n_1(n_1+1)/2$ SB's. Because each SB is atomic we may layout A on the $P \times Q$ mesh in the obvious manner. $p(I,J)$ will hold a quasi lower triangular matrix. We represent it as a one dimensional array of SB's with a column pointer (CP) array that points at the first block in $pc(J)(jl)$, $0 \leq jl < npc(J)$. Row indices are not required. This is because the last row index on each $p(I,J)$ for $0 \leq J < Q$ is the same for each I.

Although we described DMC for Symmetric/Triangular matrices it should be clear that our paradigm of using SB's works for rectangular matrices as well. Thus, most of ScaLapack's factorization codes, eg. $LU = PA$ and $QR = A$, also work under this paradigm. Our paradigm is based on the second view point of Section 2. What we have done is to isolate the major matrix multiply part of our DMC and to relate it to the Schur complement update. We also needed to add West and South border vectors to our data layout. In so doing we have eliminated the PBLAS layer. However, this is not to say that we should avoid the PBLAS layer.

Acknowledgements

The continuation part of [9], mainly Sections 2.1 and 4, as applied $LU = PA$ factorization, are recent results obtained with Sid Chatterjee, Jim Sexton and mainly John Gunnels. John implemented a Linpack benchmark that obtained 70.72 TFlops in the fall of 2004 that placed IBM number one in the TOP500 list. I thank Cleve Ashcraft for suggesting a way to write-up of Section 3.2.

References

1. R. C. Agarwal, F. G. Gustavson. A Parallel Implementation of Matrix Multiplication and LU factorization on the IBM 3090. *Proceedings of the IFIP WG 2.5 Working Group on Aspects of Computation on Asychronous Parallel Processors*, book, Margaret Wright, ed. Stanford CA. 22-26 Aug. 1988, North Holland, pp. 217-221.
2. R. C. Agarwal, F. G. Gustavson, M. Zubair. Exploiting functional parallelism of POWER2 to design high-performance numerical algorithms. *IBM Journal of Research and Development*, Vol. 38, No. 5, Sep. 1994, pp. 563–576.
3. B.S. Andersen, J. Gunnels, F. Gustavson, J. Reid, and J. Waśniewski. A fully portable high performance minimal storage hybrid format cholesky algorithm. Technical Report RAL-TR-2004-017, Rutherford Appleton Laboratory, Oxfordshire, UK and IMM-Technical Report-2004-9 `www.imm.dtu.dk/pubdb/views/publication_details.php?id=3173`, Informatics and Mathematical Modelling, Technical University of Denmark, DK-2800 Kongens Lyngby, Denmark. It is already published in: The Transaction of Mathematical Software of ACM (TOMS), vol. 31(2), pp. 201-227, 2005.
4. S. Chatterjee et. al. Design and Exploitation of a High-performance SIMD Floating-point Unit for Blue Gene/L. *IBM Journal of Research and Development*, Vol. 49, No. 2-3, March-May 2005, pp. 377-391.
5. E. Elmroth, F. G. Gustavson, B. Kagstrom, and I. Jonsson. Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software. *SIAM Review*, Vol. 46, No. 1, Mar. 2004, pp. 3,45.

6. J. A. Gunnels, F. G. Gustavson. A New Array Format for Symmetric and Triangular Matrices. This Lecture Notes in Computer Science, in section of High Performance Linear Algebra Algorithms.
7. J. A. Gunnels, F. G. Gustavson, G. M. Henry, R. A. van de Geijn. A Family of High-Performance Matrix Multiplication Algorithms. This Lecture Notes in Computer Science, in section of High Performance Linear Algebra Algorithms.
8. F. G. Gustavson. Recursion Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms. *IBM Journal of Research and Development*, Vol. 41, No. 6, Nov. 1997, pp. 737,755.
9. F. G. Gustavson High Performance Linear Algebra Algorithms using New Generalized Data Structures for Matrices. *IBM Journal of Research and Development*, Vol. 47, No. 1, Jan. 2003, pp. 31,55.
10. N. Park, B. Hong, V. K. Prasanna. Tiling, Block Data Layout, and Memory Hierarchy Performance. *IEEE Trans. Parallel and Distributed Systems*, 14(7):640-654, 2003.

Management of Deep Memory Hierarchies – Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Computations

Bo Kågström

Department of Computing Science and HPC2N
Umeå University, SE-901 87 Umeå, Sweden
bokg@cs.umu.se

Abstract. Recent progress in using recursion as a general technique for producing dense linear algebra library software for today's memory tiered computer systems is presented. To allow for efficient utilization of a memory hierarchy, our approach is to apply the technique of hierarchical blocking. The success of our approach includes novel recursive blocked algorithms, hybrid data formats and superscalar kernels.

Keywords: recursion, automatic variable blocking, superscalar, GEMM-based, level 3 BLAS, hybrid data structures, superscalar kernels, SMP parallelization, library software, LAPACK, SLICOT, ESSL, RECSY, dense linear algebra, factorizations, matrix equations, periodic systems.

1 Introduction

Matrix computations are both fundamental and ubiquitous in computational science and its vast application areas. Along with the development of more advanced computer systems with complex memory hierarchies, there is a continuing demand for new algorithms and library software that efficiently utilize and adapt to new architecture features. Since several years, our research group in Parallel and Scientific Computing at Umeå University in collaboration with IBM T.J. Watson Research Center run a successful project on the development of novel, efficient and robust algorithms and library software for the management of deep memory hierarchies. This contribution gives an introduction to our work and a brief summary of my invited plenary talk at PARA04, which presented recent advances made by applying the paradigm of recursion to dense matrix computations on today's computer systems with deep memory hierarchies (see Elmroth, Gustavson, Jonsson, and Kågström [6] and Further Readings).

1.1 Why Hierarchical Blocking Matters

Today's computers have extremely fast processors, but memory access speeds are relatively slow. Thus the performance of algorithms is frequently limited by the need to move large amounts of data between memory and the processor. This problem is particularly acute in dense matrix computations where algorithms require repeatedly sweeping through all elements of the matrix. As a result, features of computer hardware have profoundly influenced the implementation as well as the abstract description of matrix algorithms.

A key to efficient matrix computations on hierarchical memory machines is *blocking*, i.e., the matrix elements should be grouped to match the structure of a hierarchical memory system. In turn, this leads to blocked algorithms which are rich in GEMM-based level 3 operations [2,16,17].

At the top of a computer memory hierarchy are the registers where all computations (floating point, integer, and logical) take place; these have the shortest access times but the smallest storage capacities. Between the registers and main memory, there are one or more levels of cache memory with longer access time and increasing size. Closest to the registers is the first-level cache memory (L1 cache), the fastest but smallest; next are the second-level (L2) and possibly third-level (L3) cache memories. Below main memory are levels of secondary storage with larger capacity but lower access speed, such as disk and tape.

Different levels in the memory hierarchy display vastly different access times. For example, register and L1 cache access times are typically on the order of nanoseconds, whereas disk access times are in milliseconds – a difference of 10^6. Furthermore, access time changes by a factor of five or ten between each level. Thus the ideal for such a system is to perform as many calculations as possible on data that resides in the fastest cache. The storage capacity of each level also varies greatly. Registers may hold up to a few kilobytes of data, L1 cache up to a few hundred kilobytes, the highest-level cache up to a few megabytes (10^6 bytes), while today's main memory can hold several gigabytes (10^9 bytes).

2 Explicit Versus Recursive Blocking

Various strategies are known for blocking data in order to exploit a hierarchical memory structure. The classical way is *explicit multilevel blocking*, where each index/loop set matches a specific level of the memory hierarchy. This requires a detailed knowledge of the architecture and (usually) a separate blocking parameter for each level. *Register blocking* and *cache blocking* refer to variants of this idea designed for efficient reuse of data in the registers and one or more levels of cache, respectively.

The memory of most computers is laid out in blocks of fixed size, called *pages*. At any instant of computer time there is a set of “fast" pages, sometimes called the working set, which resides in the translation look-aside buffer (TLB). The term *TLB blocking* means a strategy designed so that memory is mostly accessed in the working set.

In contrast to these approaches, *recursive blocking*, which combines recursion and blocking, leads to an automatic variable blocking with the potential for matching the memory hierarchies of today's high-performance computing systems. The recursion repeatedly partitions the problem into smaller and smaller subproblems, so that data corresponding to different levels of the recursion tree fits into different levels of the memory hierarchy. Recursive blocking means that “traditional" recursion is to terminate when the size of the controlling recursion blocking parameter becomes smaller than some predefined value. This assures that all leaf computations in the recursion tree will usually be a substantial level 3 (matrix-matrix) computation on data stored in the top-level caches.

3 Recursive Blocked Algorithms

A recursive blocked algorithm is defined in terms of basic splittings that generate new smaller subproblems (tasks) in a recursion tree. For each of these subproblems a recursive template is applied, which in turn generates new tasks, etc. Here, we illustrate such templates by considering two related problems: the solution of a triangular system with multiple right hand sides (TRSM) and the solution of a continuous-time triangular Sylvester (SYCT) equation.

TRSM Operation. First, we consider solving $AX = C$, where X overwrites C. A of size $m \times m$ is upper triangular, and C and X are $m \times n$. Depending on m and n, there are several alternatives for doing a recursive splitting. Two of them are illustrated below.

Case 1 ($1 \leq m \leq n/2$). Split C by columns only,

$$A \begin{bmatrix} X_1 \; X_2 \end{bmatrix} = \begin{bmatrix} C_1 \; C_2 \end{bmatrix},$$

or, equivalently,

$$\begin{aligned} AX_1 &= C_1, \\ AX_2 &= C_2. \end{aligned}$$

Case 2 ($1 \leq n \leq m/2$). Split A, which is assumed to be upper triangular, by rows and columns. Since the number of right-hand sides n is much smaller than m, C is split by rows only,

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \end{bmatrix},$$

or, equivalently,

$$\begin{aligned} A_{11}X_1 &= C_1 - A_{12}X_2, \\ A_{22}X_2 &= C_2. \end{aligned}$$

The two splittings above are fundamental in all types of triangular solve (or multiply) operations and illustrate that a *problem is split into two subproblems with dependent and independent tasks*, respectively. In Case 1, a splitting is applied only to C, the right-hand sides, and we obtain two similar TRSM operations that can be solved independently and concurrently (illustrated in Figure 1 (left)). In Case 2, we first have to (1) solve for X_2 and (2) update the right-hand side C_1 with respect to X_2, which is a general matrix multiply and add (GEMM) operation, before (3) solving for X_1. The splitting of A imposes a *critical path* at the block level that any algorithm (recursive or nonrecursive) has to respect (illustrated in Figure 1 (right)).

There is also a Case 3 ($n/2 < m < 2n$), when all matrices involved are split by rows and columns leading to four subproblems (illustrated below for solving SYCT). Other variants of the TRSM operation ($\text{op}(A)X = C$ or $X\text{op}(A) = C$, where $\text{op}(A)$ is A or A^T, with A upper or lower triangular) are treated similarly.

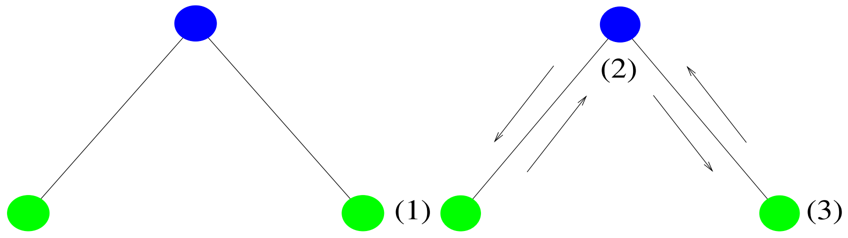


Fig. 1. Splittings defining independent tasks (left) and dependent tasks (right). The right-hand splitting defines a critical path of subtasks: (1), (2), (3)

Algorithm 1 Recursive blocked algorithm in Matlab-like code for solving the triangular continuous-time Sylvester (SYCT) equation

```
**Algorithm 1: SYCT function**[X] = **recsyct**(A,B,C,uplo,blksz)
**if** $1 \le m, n \le blksz$ **then**
    $X = \mathbf{trsyct}(A, B, C, uplo);$
**else**
    **if** $1 \le m \le n/2$   % Case 1: Split $B$ (by rows and colums), $C$ (by columns only)
      $X_1 = \mathbf{recsyct}(A, B_{11}, C_1, 1, blksz);$
      $C_2 = \mathbf{gemm}(X_1, B_{12}, C_2);$
      $X_2 = \mathbf{recsyct}(A, B_{22}, C_2, 1, blksz);$
      $X = [X_1, X_2];$
    **elseif** $1 \le n \le m/2$   % Case 2: Split $A$ (by rows and colums), $C$ (by rows only)
      $X_2 = \mathbf{recsyct}(a_{22}, B, C_2, 1, blksz);$
      $C_1 = \mathbf{gemm}(-A_{12}, X_2, C_1);$
      $X_1 = \mathbf{recsyct}(A_{11}, B, C_1, 1, blksz);$
      $X = [X_1; X_2];$
    **else**   % $m, n \ge blksz$, Case 3: Split $A$, $B$ and $C$ (all by rows and colums)
      $X_{21} = \mathbf{recsyct}(A_{22}, B_{11}, C_{21}, 1, blksz);$
      $C_{22} = \mathbf{gemm}(X_{21}, B_{12}, C_{22});\ C_{11} = \mathbf{gemm}(-A_{12}, X_{21}, C_{11});$
      $X_{22} = \mathbf{recsyct}(A_{22}, B_{22}, C_{22}, 1, blksz);\ X_{11} = \mathbf{recsyct}(A_{11}, B_{11}, C_{11}, 1, blksz);$
      $C_{12} = \mathbf{gemm}(-A_{12}, X_{22}, C_{12});$
      $C_{12} = \mathbf{gemm}(X_{11}, B_{12}, C_{12});$
      $X_{12} = \mathbf{recsyct}(A_{11}, B_{22}, C_{12}, 1, blksz);$
      $X = [X_{11}, X_{12};\ X_{21}, X_{22}];$
    **end**
**end**
```

SYCT Matrix Equation. We consider the matrix equation $AX - XB = C$, where A has size $m \times m$, B has size $n \times n$, and both are upper quasi-triangular, i.e., in real Schur form. The right-hand side C and the solution X are of size $m \times n$ and, typically, the solution overwrites the right-hand side ($C \leftarrow X$). The SYCT equation has a *unique solution* if and only if A and B have no eigenvalue in common or.

As for the TRSM operation, three alternatives for doing a *recursive splitting* are considered. In Case 1 ($1 \leq m \leq n/2$), B is split by rows and columns, and C by columns only. Similarly, in Case 2 ($1 \leq n \leq m/2$), A is split by rows and columns, and C by rows only. Finally, in Case 3 ($n/2 < m < 2n$) both rows and columns of the matrices A, B, and C are split:

$$\begin{bmatrix} A_{11} & A_{12} \\ & A_{22} \end{bmatrix} \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} - \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

This recursive splitting results in the following four triangular SYCT equations:

$$\begin{aligned} A_{11}X_{11} - X_{11}B_{11} &= C_{11} - A_{12}X_{21}, \\ A_{11}X_{12} - X_{12}B_{22} &= C_{12} - A_{12}X_{22} + X_{11}B_{12}, \\ A_{22}X_{21} - X_{21}B_{11} &= C_{21}, \\ A_{22}X_{22} - X_{22}B_{22} &= C_{22} + X_{21}B_{12}. \end{aligned}$$

Conceptually, we start by solving for X_{21} in the third equation. After updating C_{11} and C_{22} with respect to X_{21}, one can solve for X_{11} and X_{22}. Both updates and the triangular Sylvester solves are independent operations and can be executed concurrently. Finally, one updates C_{12} with respect to X_{11} and X_{22} and solves for X_{12}. In practice, all four subsystems are solved using the recursive blocked algorithm (see Algorithm 1). If a splitting point ($m/2$ or $n/2$) appears at a 2×2 diagonal block, the matrices are split just below this diagonal block. We remark that the issue of 2×2 diagonal blocks corresponding to conjugate eigenvalue pairs would infer extra overhead if a recursive data layout for matrices is used, and therefore a standard data layout is to be preferred [12].

In the discussion above, we have assumed that both A and B are upper triangular (or quasi-triangular). However, it is straightforward to derive similar recursive splittings for the triangular SYCT, where each of A and B can be in either upper or lower Schur form.

We remark that by splitting all problem dimensions simultaneously, we do a *splitting by breadth* and generate several new tasks (subproblems) at the next level of the recursion tree (four for Case 3 of SYCT). On the other hand, splitting only one of the problem dimensions generates only two new tasks (Cases 1 and 2 of SYCT), and we need to recursively repeat the splitting in order to get the same number of new tasks as when splitting by breadth. Accordingly, we call this *splitting by depth*. Typically, one makes the choice of splitting to generate "squarish" subproblems, i.e., the ratio between the number of operations made on subblocks and the number of subblocks is maintained as high as possible.

3.1 Superscalar Kernels for Leaf Computations

In order to reduce the overhead cost of recursion to a tiny and acceptable level, the recursion is terminated when the new problem sizes are smaller than a certain block size, *blksz*. The block size is chosen so that submatrices associated with a leaf computation fit in L1 cache.

Superscalar kernels are very highly performing routines working on matrix operands optimally prepared for excellent performance in L1 cache. Our work has produced designs for several superscalar kernels that are applied to the leaf nodes in the recursion tree [6]. We have found it necessary to develop such kernels, since most compilers make a poor job of generating assembler instructions that match a superscalar architecture with several registers and a long instruction pipeline. Some of the superscalar kernels make use of recursion as well [13,14]. One example is the superscalar kernel $\mathbf{trsyct}(A, B, C, uplo)$, used in the recursive blocked algorithm for the SYCT equation (see Algorithm 1).

All superscalar kernels are written in Fortran using register and cache blocking techniques such as loop unrolling. For each recursive blocked algorithm the same superscalar kernels are used on all platforms. Currently, they are optimized with a generic superscalar architecture in mind and show very good performance on several different platforms. Moreover, the generic superscalar kernels make the recursive blocked algorithms portable across different computer systems.

4 Locality Issues and Hybrid Data Structures

Recursive blocked algorithms mainly improve on the *temporal locality*, which means that blocks (submatrices) which recently have been accessed will most likely be referenced soon again. For many of our algorithms the use of the recursive blocking technique together with new superscalar kernels is enough to reach near to optimal performance.

For some problems, we can further increase the performance by explicitly improving on the *spatial locality* as well. The goal is now to match the algorithm and the data structure so that blocks (submatrices) near the recently accessed blocks will also be referenced soon. In other words, the storing of matrix blocks in memory should match the data reference pattern of the blocks, and thereby as much as possible minimize data transfers in the memory hierarchy. We use the divide-and-conquer heuristics leading to *hybrid data structures* that store the blocks recursively (e.g., see [9,10,6]). Ultimately, we want to reuse the data as much as possible at each level of the memory hierarchy and thereby minimize the cost. The combination of recursive blocking and a hybrid data format has shown to be especially rewarding in the context of packed factorizations [10].

4.1 Hybrid Packed Data Formats for Matrix Factorizations

The idea behind recursive factorization of a symmetric matrix stored in packed recursive format is simple: Given AP holding symmetric A in lower packed storage mode, overwrite AP with A in the recursive packed row format. Next, execute the recursive level 3 factorization algorithm. Here, we illustrate with the packed Cholesky factorization of a positive definite matrix [10]. The approach is applicable to most (if not all) packed matrix computations in LAPACK [2].

The packed recursive data format is a hybrid triangular format consisting of $n - 1$ full format rectangles of varying sizes and n triangles of size 1×1 on the diagonal. The format uses the same amount of data storage as the ordinary packed triangular format, i.e., $n(n + 1)/2$. Since the rectangles (square submatrices) are in full format it is possible to use high-performance level 3 BLAS on these square submatrices. The

$$\begin{array}{ccccccc}
1 & 2 & 4 & 7 & 11 & 16 & 22 \\
 & 3 & 5 & 8 & 12 & 17 & 23 \\
 & & 6 & 9 & 13 & 18 & 24 \\
 & & & 10 & 14 & 19 & 25 \\
 & & & & 15 & 20 & 26 \\
 & & & & & 21 & 27 \\
 & & & & & & 28
\end{array}
\qquad
\begin{array}{ccc|cccc|}
\cline{4-7}
1 & 2 & 3 & 7 & 10 & 13 & 16 \\
 & 4 & 5 & 8 & 11 & 14 & 17 \\
 & & 6 & 9 & 12 & 15 & 18 \\
\cline{4-7}
\multicolumn{3}{c}{} & 19 & 20 & 22 & \multicolumn{1}{c}{24} \\
\multicolumn{3}{c}{} & & 21 & 23 & \multicolumn{1}{c}{25} \\
\multicolumn{3}{c}{} & & & 26 & \multicolumn{1}{c}{27} \\
\multicolumn{3}{c}{} & & & & \multicolumn{1}{c}{28}
\end{array}$$

Packed upper Recursive packed upper

Fig. 2. Memory indices for 7×7 upper triangular matrix stored in standard packed format and recursive packed format

difference between the standard packed and the recursive packed formats is shown in Figure 2 for a matrix of order 7.

Notice that the triangles are split into two triangles of sizes $n_1 = n/2$ and $n_2 = n - n_1$ and a rectangle of size $n_2 \times n_1$ for lower format and $n_1 \times n_2$ for upper format. The elements in the upper left triangle are stored first, the elements in the rectangle follows, and the elements in the lower right triangle are stored last. The order of the elements in each triangle is again determined by the recursive scheme of dividing the sides n_1 and n_2 by two and ordering these sets of points in the order triangle, rectangle, triangle. The elements in the rectangle are stored in full format, either by row or by column. The new recursive packed format was first presented in [1] and is based on the formats described in [9]. An algorithm which transforms from conventional packed format to recursive packed format can be found in [10].

The recursive formulation of the algorithm is straightforwardly derived from the block factorization of a positive definite matrix A:

$$A \equiv \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = LL^T \equiv \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22} \end{bmatrix},$$

which consists of two Cholesky factorizations (4.1), (4.4), one triangular system solve with multiple right-hand sides (4.2), and one symmetric rank-k update (4.3),

$$A_{11} = L_{11}L_{11}^T, \tag{4.1}$$
$$L_{21}L_{11}^T = A_{21}, \tag{4.2}$$
$$\tilde{A}_{22} = A_{22} - L_{21}L_{21}^T, \tag{4.3}$$
$$\tilde{A}_{22} = L_{22}L_{22}^T. \tag{4.4}$$

These equations build the *recursive template* for the recursive Cholesky factorization. After recursively solving for L_{11}, a recursive implementation of TRSM is used to solve for L_{21}. Then a recursive symmetric rank-k update (SYRK) of A_{22} occurs before L_{22} is recursively Cholesky factored. The need of the recursive TRSM and SYRK stems from the recursive packed format. The factorization algorithm calls TRSM and SYRK with triangular matrix operands stored in recursive packed format, and with rectangular matrix operands stored in full format. Dividing the recursive packed matrices in TRSM and SYRK gives rise to two recursive packed triangular matrices and a rectangular matrix stored in full format, which becomes an argument to GEMM. The recursive blocking in the Cholesky algorithm (as well as in the recursive TRSM and SYRK) is only used down to a fixed block size when superscalar kernels are used to solve the respective leaf problems.

The recursive algorithm for Cholesky factorization has several attractive features. First, it uses minimal storage. Second, it attains level 3 performance due to mostly performing GEMM operations during execution. In addition, it outperforms the LAPACK routine DPPTRF for standard packed data storage up to a factor 3.5 (IBM Power3, 200 MHz)[10]. The main reason is that the standard approach for packed storage of matrices (typified by LAPACK [2]) cannot use standard level 3 BLAS subroutines. Notably, even when one includes the cost of converting the data from conventional packed to recursive packed format, the performance turns out to be better than LAPACK's level 3 routine DPOTRF for full storage format.

5 RECSY Library

RECSY [15] is a high-performance library for solving triangular Sylvester-type matrix equations. The Fortan 90 routines are implementations of the recursive blocked algorithms presented in [13,14] for one-sided and two-sided matrix equations (see Table 1). The classification in one-sided and two-sided matrix equations distinguishes the type of matrix product terms that appear and the way updates are performed in the recursive blocked algorithms. *One-sided* matrix equations include terms where the solution is only involved in matrix products of two matrices, e.g., op$(A)X$ or Xop(A), where op(A) can be A or A^T. The SYCT equation discussed earlier is one of them. *Two-sided* matrix equations include matrix product terms of type op$(A)X$op(B), and examples are the discrete-time standard and generalized Sylvester and Lyapunov equations.

In total, 42 different cases of eight equations (three one-sided and five two-sided) are solved by the library, either in serial or in parallel using OpenMP. The library includes superscalar kernels, much faster than traditional SLICOT [18] or LAPACK kernels. The

Table 1. One-sided (top) and two-sided (bottom) matrix equations. (CT: continuous-time, DT: discrete-time)

Name	Matrix equation	Acronym
Standard Sylvester (CT)	$AX - XB = C$	SYCT
Standard Lyapunov (CT)	$AX + XA^T = C$	LYCT
Generalized Coupled Sylvester	$(AX - YB, DX - YE) = (C, F)$	GCSY
Standard Sylvester (DT)	$AXB^T - X = C$	SYDT
Standard Lyapunov (DT)	$AXA^T - X = C$	LYDT
Generalized Sylvester	$AXB^T - CXD^T = E$	GSYL
Generalized Lyapunov (CT)	$AXE^T + EXA^T = C$	GLYCT
Generalized Lyapunov (DT)	$AXA^T - EXE^T = C$	GLYDT

new kernels do not overflow and provide near-singularity checking. If the problem is ill-conditioned, the routine as an alternative backtracks and uses a kernel that carries out complete pivoting in the solution of small-sized leaf problems of the recursion tree.

In order to make the library easy to use, wrapper routines for SLICOT and LAPACK are included, so the user can keep his/her original code and simply link with RECSY. This means that the user calls the SLICOT routine for solving an unreduced problem, and the transformed quasi-triangular matrix equation is automatically solved by RECSY.

For illustration, we show a few results obtained with the RECSY library. In all cases, $m = n$. In Table 2 a), results obtained for the continuous-time Lyapunov (LYCT) equation are shown. Here, the speedup is remarkable. The RECSY library is up to 83 times faster than the original SLICOT library. This is both due to faster kernels and the automatic variable-sized multi-level blocking from recursion. Similar results can be observed for LAPACK routines. For example, the RECSY routine RECSYCT is more than 20 times faster then the LAPACK routine DTRSYL for $M = N > 500$ on the IBM PowerPC 604e. Timings for two-sided matrix equation examples are given in Table 2 b). For the largest example, the SLICOT library requires more than 45 minutes to solve the problem. The RECSY library solves the same problem in less than 30 seconds. The extra speedup from the OpenMP version of the library is also given. For further results, we refer to [13,14] and the RECSY homepage [15].

Table 2. a) Performance results for the triangular Lyapunov equation—IBM Power3, 200 MHz (left) and SGI Onyx2 MIPS R10000, 195 MHz (right). b) Performance results for the triangular discrete-time Sylvester (SYDT) equation—IBM Power3, 4 × 375 MHz

	a) $AX + XA^T = C$						b) $AXB - X = C$			
	IBM Power3			MIPS R10000			IBM Power3			
	Mflops/s		Speedup	Mflops/s		Speedup	Time (sec)		Speedup	
m	A	B	B/A	A	B	B/A	C	D	D/C	E/D
100	77.0	166.5	2.16	82.0	123.5	1.51	1.73e-2	7.41e-3	2.33	1.16
250	85.3	344.5	4.04	88.7	224.5	2.53	6.20e-1	6.93e-2	8.95	0.98
500	10.6	465.0	43.85	42.2	277.8	6.58	2.32e+1	4.60e-1	50.50	1.48
1000	7.7	554.7	72.20	14.5	254.0	17.57	2.44e+2	3.26e+0	74.65	1.94
1500	7.0	580.5	83.19	9.7	251.0	25.81	9.37e+2	1.08e+1	86.66	2.04
2000							2.84e+3	2.41e+1	117.71	2.14
	A – SLICOT SB03MY B – RECLYCT						C – SLICOT SB04PY D – RECSYDT E – RECSYDT_P			

6 Recursion in Matrix Factorizations and Linear Systems

It is straigthforward to design recursive blocked algorihms for one-sided matrix factorizations (typified by Cholesky, LU, and QR) [8,6]. Our generic recursive template looks as follows:

Recursively factor A:

1. *Partition*

$$A \equiv \begin{bmatrix} A_1 & A_2 \end{bmatrix}, \quad \text{where } A_1 \equiv \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} \text{ and } A_2 \equiv \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}.$$

2. *Recursively factor* A_1 (or A_{11}).
3. *Apply* resulting transformations to $A_2 \equiv \left[\begin{smallmatrix} A_{12} \\ A_{22} \end{smallmatrix}\right]$.
4. *Recursively factor* A_{22}.

In the algorithm, the template is recursively applied to the two smaller subproblems (factorizations in steps 2 and 4). As for the triangular solve operation, the recursion template results in a splitting defining a critical path of dependent subtasks (see Figure 1 (right)).

Recent work includes recursive blocked algorithms for the QR factorization [3,5] and for solving general linear systems $AX = B$ [4], where both over- and underdetermined linear systems are considered. One important message is that various level 2 and level 3 algorithms in LAPACK can be successfully replaced by recursive level 3 algorithms.

7 Summary and Future Work

The SIAM Review article [6] gives a survey of recent progress in using recursion as a general technique for producing dense linear algebra software that is efficient on today's memory-tiered computers. In addition to general ideas and techniques, detailed case studies of matrix computations are presented, which only briefly have been dicussed here. Some of the main points are the following:

– Recursion creates new algorithms for linear algebra software.
– Recursion can be used to express dense linear algebra algorithms entirely in terms of level 3 BLAS-like matrix-matrix operations.
– Recursion introduces an automatic variable blocking that targets every level of a deep memory hierarchy.
– Recursive blocking can also be used to define data formats for storing block-partitioned matrices. These formats generalize standard matrix formats for general matrices, and generalize both standard packed and full matrix formats for triangular and symmetric matrices.

In addition, new algorithms and library software for level 3 BLAS [16,17,9], matrix factorizations [3,5,8,10], the solution of general linear systems [4], and common matrix equations [13,14,15,12,7] are described. Some software implementations are included in the IBM ESSL library [11], including factorizations and solvers for positive definite systems as well as the QR factorization and solution of general over- and underdetermined systems. All software implementations of the triangular matrix equations and condition estimators are available in the RECSY library [15]. The performance results of our software are close to (i.e., within 50% to 90% of) the peak attainable performance of the machines for large enough problems, showing the effectiveness of the techniques described.

Based on our successful work on using recursion in solving different types of matrix equations (e.g., see [13,14,15,12,7]), ongoing work includes solving periodic Sylvester-type equations. This leads to recursive blocked algorithms for 3-dimensional data structures.

Furthermore, we will investigate open problems in how to apply recursion to improve performance of "two-sided" linear algebra operations, such as the reduction to (generalized) Hessenberg, symmetric tridiagonal, or bidiagonal forms. One challenge is to see if recursion can reduce the nontrivial fraction of level 1 and level 2 operations that are currently applied to both sides of the matrix under reduction.

Further Readings and Acknowledgements

Most of the work presented here represents joint work with Erik Elmroth, Fred Gustavson and Isak Jonsson. For further information about our work, see the SIAM Review article [6] and the selection of references listed below. In addition, see the comprehensive list of references in [6] on related and complementary work. Thanks to you all!

This research was conducted using the resources of the High Performance Computing Center North (HPC2N). Financial support has been provided by the *Swedish Research Council* under grant VR 621-2001-3284 and by the *Swedish Foundation for Strategic Research* under the frame program grant SSF-A3 02:128.

References

1. B. Andersen, F. Gustavson, and J. Waśniewski, A recursive formulation of Cholesky factorization of a matrix in packed storage, *ACM Trans. Math. Software*, 27 (2001), pp. 214–244.
2. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, 1999.
3. E. Elmroth and F. G. Gustavson, Applying recursion to serial and parallel QR factorization leads to better performance, *IBM J. Res. Develop.*, 44 (2000), pp. 605–624.
4. E. Elmroth and F. G. Gustavson, A faster and simpler recursive algorithm for the LAPACK routine DGELS, *BIT*, 41 (2001), pp. 936–949.
5. E. Elmroth and F. G. Gustavson, High-performance library software for QR factorization, in *Applied Parallel Computing: New Paradigms for HPC in Industry and Academia*, T. Sørvik et al., eds., *Lecture Notes in Comput. Sci.* 1947, Springer-Verlag, New York, 2001, pp. 53–63.
6. E. Elmroth, F. Gustavson, I. Jonsson, and B. Kågström, Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software, *SIAM Review*, Vol. 46, No. 1, 2004, pp. 3–45.
7. R. Granat, I. Jonsson, and B. Kågström, Combining Explicit and Recursive Blocking for Solving Triangular Sylvester-Type Matrix Equations on Distributed Memory Platforms, in *Euro-Par 2004 Parallel Processing*, M. Danelutto, D. Laforenza, and M. Vanneschi, eds., *Lecture Notes in Comput. Sci.* 3149, Springer-Verlag, Berlin Heidelberg, 2004, pp. 742–750.

8. F. G. Gustavson, Recursion leads to automatic variable blocking for dense linear-algebra algorithms, *IBM J. Res. Develop.*, 41 (1997), pp. 737–755.
9. F. G. Gustavson, A. Henriksson, I. Jonsson, B. Kågström, and P. Ling, Recursive blocked data formats and BLAS's for dense linear algebra algorithms, in *Applied Parallel Computing: Large Scale Scientific and Industrial Problems*, B. Kågström et al., eds., *Lecture Notes in Comput. Sci.* 1541, Springer-Verlag, New York, 1998, pp. 195–206.
10. F. G. Gustavson and I. Jonsson, Minimal-storage high-performance Cholesky factorization via blocking and recursion, *IBM J. Res. Develop.*, 44 (2000), pp. 823–849.
11. IBM, *Engineering and Scientific Subroutine Library, Guide and Reference*, Ver. 3, Rel. 3, 2001.
12. I. Jonsson, Analysis of Processor and Memory Utilization of Recursive Algorithms for Sylvester-Type Matrix Equations Using Performance Monitoring, Report UMINF-03.16, Dept. of Computing Science, Umeå University, Sweden, 2003.
13. I. Jonsson and B. Kågström, Recursive blocked algorithms for solving triangular systems—Part I: One-sided and coupled Sylvester-type matrix equations, *ACM Trans. Math. Software*, 28 (2002), pp. 392–415.
14. I. Jonsson and B. Kågström, Recursive blocked algorithms for solving triangular systems—Part II: Two-sided and generalized Sylvester and Lyapunov equations, *ACM Trans. Math. Software*, 28 (2002), pp. 416–435.
15. I. Jonsson and B. Kågström, RECSY—A High Performance Library for Sylvester-Type Matrix Equations, http://www.cs.umu.se/research/parallel/recsy, 2003.
16. B. Kågström, P. Ling, and C. Van Loan. GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Software*, 24(3):268–302, 1998.
17. B. Kågström, P. Ling, and C. Van Loan, Algorithm 784: GEMM-based level 3 BLAS: Portability and optimization issues, *ACM Trans. Math. Software*, 24 (1998), pp. 303–316.
18. SLICOT, *The SLICOT Library and the Numerics in Control Network (NICONET) website*, http://www.win.tue.nl/niconet/.

Fortran Is Getting More and More Powerful

John K. Reid*

Atlas Centre, Rutherford Appleton Laboratory, UK
j.k.reid@rl.ac.uk

Abstract. There is plenty happening just now with respect to Fortran.
Two sets of features (for exception handling and for enhancements to allocatable arrays) were defined in Technical Reports[1] as extensions to Fortran 95 and have become widely available in compilers.
The Fortran 2003 Standard has been approved and is about to be published. As well as adding the contents of the two Technical Reports, this adds interoperability with C, parameterized derived types, procedure pointers, type extension and polymorphism, access to the computing environment, support of international character sets, and many other enhancements.
A new Technical Report has also been approved and is about to be published. This enhances the module features and avoids the 'compilation cascade' that can mar the development of very large programs. It is written as an extension of Fortran 2003, but is expected to be widely implemented as an extension to Fortran 95 compilers.
We will summarize all these developments, which will make Fortran even more suitable for large numerically-demanding applications.

1 Introduction and Overview of the New Features

Fortran is a computer language for scientific and technical programming that is tailored for efficient run-time execution on a wide variety of processors. It was first standardized in 1966 and the standard has since been revised three times (1978, 1991, 1997). The revision of 1991 was major and those of 1978 and 1997 were relatively minor. The fourth revision is major and has been made following a meeting of ISO/IEC JTC1/SC22/WG5 in 1997 that considered all the requirements of users, as expressed through their national bodies. At the time of writing (September 2004), it has passed its DIS (Draft International Standard) ballot and is about to be published as a Standard. The DIS is visible as WG5 document N1601 at `ftp://ftp.nag.co.uk/sc22wg5/N1601-N1650/`.

Two features, allocatable array extensions and support for the exceptions of the IEEE Floating Point Standard[1] were not ready in time for Fortran 95, but were deemed too important to wait for the next revision, so were defined in Technical Reports with a promise that they would be included in the revision. Many Fortran 95 compilers implement them as extensions.

Late in the revision process, a remedy was suggested for a significant deficiency in the module feature for very large programs. Rather than risk a delay to the Standard, it was decided that this should be another Technical Report. It has now caught up

* Convener, ISO Fortran Committee

[1] A mini Standard with a simpler and quicker ratification process.

with the Standard, has passed its DTR (Draft Technical Report) ballot, and is about to be published as a Technical Report. The DTR is visible as document N1602 at `ftp://ftp.nag.co.uk/sc22wg5/N1601-N1650/`.

The significant enhancements in the 1991 revision were dynamic storage, structures, derived types, pointers, type parameterization, modules, and array language. The main thrust of the 1997 revision was in connection with alignment with HPF (High Performance Fortran). The major enhancements for this revision and its associated Technical Report are:

1. Allocatable array extensions.
2. Support for the exceptions of the IEEE Floating Point Standard [1].
3. Enhanced module facilities.
4. Interoperability with the C programming language.
5. Data enhancements and object orientation: parameterized derived types, type extension and inheritance, procedure pointers, polymorphism, dynamic type allocation, and type-bound procedures.
6. Input/output enhancements: asynchronous transfer, stream access, user specified transfer operations for derived types, and forms for IEEE Nans and infinities.
7. Miscellaneous enhancements: improved structure constructors, improved allocate features, access to ISO 10646 4-byte characters, and access to command line arguments and environment variables.

Each of these are described briefly in the sections that follow. It is by no means a complete description of the whole language. For fuller descriptions, see [3], [4], or [5].

Except in extremely minor ways, this revision is upwards compatible with the current standard, that is, a program that conforms to the present standard will conform to the revised standard.

The enhancements are in response to demands from users and will keep Fortran appropriate for the needs of present-day programmers without losing the vast investment in existing programs.

2 Allocatable Array Extensions

Fortran 2003, and Technical Report ISO/IEC 15581 TR:2001(E), allows the use of allocatable arrays as dummy arguments, function results, and components of structures. Pointer arrays may be used instead, but there are significant advantages in using allocatable arrays:

1. Code for a pointer array is likely to be less efficient because allowance has to be made for strides other than unity. For example, its target might be the section `vector(1:n:2)`.
2. Intrinsic assignment is often unsuitable for a derived type with a pointer component because the assignment

   ```
   a = b
   ```

   will leave `a` and `b` sharing the same target for their pointer component. If `a` and `b` are of a type with an allocatable component `c`, the effect is:

   (a) if `a%c` is allocated, it is deallocated;
   (b) if `b%c` is allocated, `a%c` is allocated with its size and is given its value.
3. If a defined operation involves a temporary variable of a derived type with a pointer component, the compiler will probably be unable to deallocate its target when storage for the variable is freed, that is, memory will 'leak'. Consider, for example, the statement

   ```
   a = b + c*d    ! a, b, c, and d
                  ! are of the same derived type
   ```

   This will create a temporary for `c*d`, which is not needed once `b + c*d` has been calculated.
4. Similar considerations apply to a function invocation within an expression. The compiler will be unlikely to be able to deallocate the pointer after the expression has been calculated and memory will leak.

3 Floating-Point Exception Handling

Most computers nowadays have hardware based on the IEEE standard for binary floating-point arithmetic [1]. Therefore, the exception handling features of Fortran 2003, and Technical Report ISO/IEC 15580 TR:2001(E), are based on the ability to test and set the five flags for floating-point exceptions that the IEEE standard specifies. However, non-IEEE computers have not been ignored; they may provide support for some of the features and the programmer is able to find out what is supported or state that certain features are essential.

Few (if any) computers support every detail of the IEEE standard. This is because considerable economies in construction and increases in execution performance are available by omitting support for features deemed to be necessary to few programmers. It was therefore decided to include inquiry facilities for the extent of support of the standard, and for the programmer to be able to state which features are essential. For example, the inquiry functions

```
ieee_support_inf(x)
ieee_support_nan(x)
```

return true or false according to whether or not IEEE infinities and IEEE NaNs are supported for the reals of the same kind as `x`.

The mechanism finally chosen is based on a set of procedures for setting and testing the flags and inquiring about the features, collected in an intrinsic module called `ieee_exceptions`. For example, the elemental subroutines:

```
ieee_get_flag(flag,flag_value)
ieee_set_flag(flag,flag_value)
```

get or set one or more flags.

Given that procedures were being provided for the IEEE flags, it seemed sensible to provide procedures for other aspects of the IEEE standard. These are collected in a separate intrinsic module, `ieee_arithmetic`, which contains a `use` statement for `ieee_exceptions`. For example, the elemental function

```
ieee_is_nan(x)
```

returns true or false according to whether or not the value of x is a NaN. If x is an array, it returns a logical array of the same shape as x. Another example is that the subroutines

```
ieee_get_rounding_mode(round_value)
ieee_set_rounding_mode(round_value)
```

allow the current rounding mode to be determined or changed.

To provide control over which features are essential, there is a third intrinsic module, ieee_features, containing named constants corresponding to the features. If a named constant is accessible in a scoping unit, the corresponding feature must be available there. For example, the statement:

```
use, intrinsic :: ieee_features, &
                  only:ieee_invalid_flag
```

ensures that the IEEE invalid flag is supported.

4 Enhanced Module Facilities

The module facility is useful for structuring Fortran programs because it allows related procedures, data, and types to be collected into a single coherent program unit. This facility is adequate for small and moderate-size programs, but has deficiencies for large programs or programs with large modules. The slightest change in a module can lead to the recompilation of every program unit that uses the module, directly or indirectly (a 'compilation cascade').

Technical Report ISO/IEC 19767 addresses these deficiencies by allowing the implementation parts of a module to be defined separately in submodules. The compilation of program units that use a module depends only on its interface parts, so this is not affected by the development and maintenance of its submodules. This also provides benefits for packaging proprietary software since the module may be published openly while the submodules remain proprietary.

The provisions are posed as an amendment to Fortran 2003, and are compatible with it. They are also compatible with Fortran 95 and we expect vendors to implement them as an extension of Fortran 95.

5 Interoperability with C

Fortran 2003 provides a standardized mechanism for interoperating with C. Clearly, any entity involved must be such that equivalent declarations of it may be made in the two languages. This is enforced within the Fortran program by requiring all such entities to be 'interoperable'. We will explain in turn what this requires for types, variables, and procedures. They are all requirements on the syntax so that the compiler knows at compile time whether an entity is interoperable.

There is an intrinsic module named iso_c_binding that contains named constants holding kind type parameter values for many intrinsic C types, for example c_float for C's `float`. The processor is not required to support all of them. Lack of support is indicated with a negative value.

For a derived type to be interoperable, it must be given the `bind` attribute explicitly and each component must be interoperable and must not be a pointer or allocatable. This allows Fortran and C types to correspond.

A scalar Fortran variable is interoperable if it is of interoperable type and is neither a pointer nor allocatable.

An array Fortran variable is interoperable if it is of interoperable type, and is of explicit shape or assumed size. It interoperates with a C array of the same type, type parameters and shape, but with reversal of subscripts. For example, a Fortran array declared as

```
integer :: a(18, 3:7, *)
```

is interoperable with a C array declared as

```
int b[][5][18]
```

A new attribute, `value`, has been introduced for scalar dummy arguments. When the procedure is called, a copy of the actual argument is made. The dummy argument is a variable that may be altered during execution of the procedure, but no copy back takes place on return.

A Fortran procedure is interoperable if it has an explicit interface and is declared with the `bind` attribute:

```
function func(i,j,k,l,m), bind(c)
```

All the dummy arguments must be interoperable. The procedure has a 'binding label', which has global scope and is the name by which it is known to the C processor. By default, it is the lower-case version of the Fortran name, but an alternative may be specified explicitly:

```
function func(i,j,k,l,m), bind(c,name='c_func')
```

For interoperating with C pointers (which are just addresses), the module contains the derived types `c_ptr` and `c_funptr` that are interoperable with C object and function pointer types, respectively. There are named constants for the corresponding null values of C. There are procedures for returning the C addresses of a Fortran object or procedure and for constructing a Fortran pointer from a C pointer.

6 Data Enhancements and Object Orientation

An obvious deficiency of Fortran 95 is that whereas each of the intrinsic types has a kind parameter and character type has a length parameter, it is not possible to define a derived type that is similarly parameterized. This deficiency is remedied with a very flexible facility that allows any number of 'kind' and 'length' parameters. A kind parameter is a constant (fixed at compile time) and may be used for a kind parameter of a component of intrinsic (or derived) type. A length parameter is modelled on the length parameter for type character and may be used for declaring character lengths of character components and bounds of array components. For example, the code

```
type matrix(kind,m,n)
   integer, kind :: kind
   integer, len :: m,n
   real(kind) :: element(m,n)
end type
```

declares a type for rectangular matrices and the code

```
type(matrix(kind(0.0d0),10,20)) :: a
write(*,*) a%kind, a%m, a%n
```

declares an object of this type for holding a 10×20 double precision matrix. The syntax for component selection, for example, `a%kind` and `a%m`, is used for accessing the new type parameters.

A pointer or pointer component may be a procedure pointer. It may have an explicit or implicit interface and its association with a target is as for a dummy procedure, so its interface is not permitted to be generic or elemental. For example

```
procedure(proc), pointer :: p
procedure(), pointer     :: q
procedure(real), pointer :: r
p => fun
```

declares three procedure pointers and associates `p` with the procedure `fun`. The pointer `p` has the interface of `proc` (and so must `fun`). The pointers `q` and `r` have implicit interfaces; `r` can point only to a real function.

Having procedure pointers fills a big hole in the Fortran 95 language. It permits ‘methods’ to be carried along with objects (dynamic binding).

A procedure may be bound to a type and accessed by component selection syntax from a scalar object of the type rather as if it were a procedure pointer component with a fixed target:

```
type t
  : ! Component declarations
contains
  procedure :: proc => my_proc
end type t
type(t) :: a
    :
call a%proc(x,y)
```

A procedure may also be bound to a type as an operator or a defined assignment. In this case, the operation is accessible wherever an object of the type is accessible.

A derived type may be extended, that is, a new type may be declared that has all the components of an existing type and some additional ones. For example, the above matrix type might be extended to a type that also holds its factorization:

```
type, extends(matrix) :: factored_matrix
   logical :: factored=.false.
   real(matrix%kind) :: factors(matrix%m,matrix%n)
end type
```

The feature allows code to be written for objects of a given type and used later for objects of an extended type.

The `associate` construct associates named entities with expressions or variables during the execution of its block:

```
associate ( z=>exp(-(x**2+y**2)) )
   print *, a+z, a-z
end associate
```

The named entity takes its properties from the expression or variable.

An entity may declared with the `class` keyword in place of the `type` keyword:

```
class (matrix(kind(0.0),10)) :: f
```

It is then **polymorphic** and is able during execution to take its declared type or any of its extensions. For example, the type of an actual argument may be an extension of the type of the corresponding dummy argument.

Access to the extended parts is available through the `select type` construct, which selects for execution at most one of its constituent blocks, depending on the dynamic type of a given variable or expression:

```
select type (f)
   type is (matrix)
     : ! block of statements
   class is (factored_matrix)
     : ! block of statements
   class default
end select
```

An object may be declared as **unlimited polymorphic**

```
class (*) :: upoly
```

so that any extensible type extends it. Its declared type is regarded as different from that of any other entity.

The allocatable attribute is no longer restricted to arrays and a source variable may be specified to provide values for deferred-length type parameters and an initial value for the object itself. For example,

```
type(matrix(wp,m=10,n=20)) :: a
type(matrix(wp,m=:,n=:)),  allocatable:: b
   :
allocate(b,source=a)
```

allocates the scalar object `b` to be 10×20 matrix with the value of `a`. Alternatively, the type and type parameters may be specified directly:

```
    allocate( type(matrix(wp,m=10,n=20)) :: b )
```

For a polymorphic allocatable object, one or other of these forms must be used in order to specify the dynamic type.

7 Input/Output Enhancements

Fortran 95 has facilities for defining operators for derived types, but nothing for defining i/o processing. In Fortran 2003, it may be arranged that when a derived-type object is encountered in an input/output list, a Fortran subroutine of one of the forms

```
    subroutine formatted_io (dtv,unit,iotype,v_list, &
                             iostat,iomsg)
    subroutine unformatted_io(dtv,unit,iostat,iomsg)
```

is called. This reads some data from the file and constructs a value of the derived type or accepts a value of the derived type and writes some data to the file. In the formatted case, the `dt` edit descriptor passes a character string and an integer array to control the action. An example is

```
    character(*) :: format = &
                    "(dt 'linked-list' (10, -4, 2))"
      :
    write(unit,format) a
```

Here the user's subroutine receives the structure `a`, the unit onto which to write its value, the string `linked-list` and an integer array with elements 10, -4, 2. This allows it to perform the appropriate output.

Input/output may be asynchronous, that is, other statements may execute while an input/output statement is in execution. It is permitted only for external files opened with `asynchronous='yes'` in the `open` statement and is indicated by `asynchronous ='yes'` in the `read` or `write` statement. Execution of an asynchronous input/output statement initiates a 'pending' input/output operation, which is terminated by an explicit wait operation for the file:

```
    wait(10)
```

or implicitly by an inquire, a `close`, or a file positioning statement for the file.

Stream access is a new method of accessing an external file. It is established by specifying `access='stream'` on the `open` statement and may be formatted or unformatted. The file is positioned by 'file storage units', normally bytes, starting at position 1. The current position may be determined from a `pos=` specifier in an `inquire` statement for the unit. A required position may be indicated in a `read` or `write` statement by a `pos=` specifier.

Input and output of IEEE infinities and NaNs, now done in a variety of ways as extensions of Fortran 95, is specified. All the edit descriptors for reals treat these values in the same way and only the field width `w` is taken into account. The output forms, each right justified in its field, are:

1. `-Inf` or `-Infinity` for minus infinity
2. `Inf`, `+Inf`, `Infinity`, or `+Infinity` for plus infinity
3. `NaN`, optionally followed by non-blank characters in parentheses (to hold additional information).

8 Miscellaneous Enhancements

The value list of a structure constructor may use the syntax of an actual argument list with keywords that are component names. Components that were given default values in the type definition may be omitted.

Individual components of a derived type may be given the `private` or `public` attribute. The `protected` attribute for a variable declared in a module has been introduced to specify that it may be altered only within the module itself.

Assignment to an allocatable array is treated in the same way as assignment to an allocatable component (see Section 2): the destination array is allocated to the correct shape if it is unallocated or reallocated to the correct shape if it is allocated with another shape.

The intrinsic subroutine `move_alloc(from,to)` has been introduced to move an allocation from one allocatable object to another.

Pointer assignment for arrays has been extended to allow lower bounds to be specified and the elements of a rank-one array to be mapped:

```
p(0:,0:) => a
q(1:m,1:2*m) => a(1:2*m*m)
```

Fortran 95 did not permit `intent` to be specified for pointer dummy arguments because of the ambiguity of whether it should refer to the pointer association status, the value of the target, or both. It is permitted in Fortran 2003 and refers to the pointer association status.

Intrinsic procedures have been added to provide access to the command line and to environment variables.

Fortran 90 introduced the possibility of multi-byte character sets, which provides a foundation for supporting ISO 10646 [2]. This is a standard for 4-byte characters, which is wide enough to support all the world's languages. There is an inquiry function for the `kind` value and support for the standardized method (UTF-8) of representing 4-byte characters as strings of 1-byte characters in a file.

Names of length up to 63 characters and statements of up to 256 lines are allowed. The main reason for the longer names and statements is to support the requirements of codes generated automatically.

A binary, octal or hex constant is permitted as a principal argument in a call of the intrinsic function `int`, `real`, `cmplx`, or `dble`:

```
i = int(o'345')
r = real(z'1234ABCD')
```

For `int`, the 'boz' constant is treated as if it were an integer constant. For the others, it is treated as having the value that a variable of the type and kind would have if its value was the bit pattern specified.

Square brackets are permitted as an alternative to `(/` and `/)` as delimiters for an array constructor.

9 Conclusions

We have attempted to give a overview of a major revision of the Fortran language. It has been designed carefully to preserve major advantages of Fortran: relative ease for writing codes that run fast and strength for processing of arrays. This has led to a conservative approach for object orientation: no multiple inheritance and a clear distinction between polymorphic and non-polymorphic objects. Similarly, for interfacing with C, the temptation to import most of the C language into Fortran has been resisted.

Which of the new features excites you most is a personal matter: it may be standardized interfacing with C, exception handling, object orientation, allocatable components, or being able to access the command line. Certainly, there is something for you - Fortran will be a more powerful language.

References

1. IEEE. Binary floating-point arithmetic for microprocessor Systems. *IEC 60559: 1989*. Originally IEEE 754-1985.
2. ISO 10646. Universal multiple-octet coded character set (UCS) - Part 1: Architecture and basic multilingual plane. *ISO/IEC 10646-1:2000.*
3. Michael Metcalf, John Reid, and Malcolm Cohen. Fortran 95/2003 explained. *Oxford University Press, 2004.*
4. John Reid. The new features of Fortran 2003. *ISO/IEC JTC1/SC22/WG5 N1579, 2003.* `ftp://ftp.nag.co.uk/sc22wg5/N1551-N1600/`
5. John Reid. The future of Fortran. *Computing in Science and Engineering, 5, 4, 59-67, July/August 2003.*

Large-Scale Computations with the Unified Danish Eulerian Model

Zahari Zlatev

National Environmental Research Institute
Frederiksborgvej 399, P. O. Box 358, DK-4000 Roskilde, Denmark
zz@dmu.dk
www.dmu.dk/AtmosphericEnvironment/staff/zlatev.htm

Abstract. The Unified Danish Eulerian Model (UNI-DEM) is an mathematical model for performing different comprehensive studies related to damaging effects from high pollution levels in Denmark and Europe. The model is described with a system of partial differential equations (PDEs). The number of equations is equal to the number of chemical species that are studied by the model. Three chemical schemes (with 35, 56 and 168 chemical species) are available at present. The model can be run either as 1-layer model (a 2-D model) or as 10 layer model (a 3-D model). Three grids can be specified in the horizontal planes: (i) a coarse 96×96 grid (corresponding to $50\ km \times 50\ km$ grid-squares), (ii) a medium 288×288 grid (corresponding to $16.67\ km \times 16.67\ km$ grid-squares), (iii) a fine 480×480 grid (corresponding to $10\ km \times 10\ km$ grid-squares).
The application of some splitting procedure followed by a discretization of the spatial derivatives leads to the solution of several systems of ordinary differential equations (ODEs) at each time-step. The number of equations in each system of ODEs is equal to the product of the number of grid-points and the number of chemical species. If the 480×480 horizontal grid is used in the 3-D model with 168 chemical species, then the number of equations in each system of ODEs is equal to $N = 480 \times 480 \times 10 \times 168 = 387072000$. The number of time-steps for a run covering meteorological data for one year is 213 120 (corresponding to a time-stepsize of 150 s). Such huge computational tasks can be treated only if (i) fast numerical methods are selected, (ii) the code is optimized for running on computers with multi-hierarchical memories (i.e. if the caches are properly exploited) and (iii) parallel computers are efficiently used. The success achieved in the efforts to satisfy these three conditions and to carry out long-term computations with UNI-DEM will be reported in this paper. Studies that are related to different important topics have been performed by using the model. The most important of these studies are listed in the end of the paper.

1 Why Are Large-Scale Mathematical Models Used?

The control of the pollution levels in different highly polluted regions of Europe and North America (as well as in other highly industrialized parts of the world) is an important task for the modern society. Its relevance has been steadily increasing during the last two-three decades. The need to establish reliable control strategies for the air pollution levels will become even more important in the future. Large-scale air pollution models can successfully be used to design reliable control strategies. Many different tasks have to be solved before starting to run operationally an air pollution model. The following tasks are most important:

- describe in an adequate way all important physical and chemical processes,
- apply fast and sufficiently accurate numerical methods in the different parts of the model,
- ensure that the model runs efficiently on modern high-speed computers (and, first and foremost, on different types of parallel computers),
- use high quality input data (both meteorological data and emission data) in the runs,
- verify the model results by comparing them with reliable measurements taken in different parts of the space domain of the model,
- carry out some sensitivity experiments to check the response of the model to changes of different key parameters

and

- visualize and animate the output results to make them easily understandable also for non-specialists.

The solution of the first three tasks will be the main topic of this paper. However, different kinds of visualizations have been used to present results from some real-life runs for the studies that are listed in the end of the paper. The air pollution model, which is actually used here, is the Danish Eulerian Model (DEM); see [2], [5], [7]. In fact, a new version of this model, the Unified Danish Eulerian Model (UNI-DEM), has recently been developed and this new version will mainly be used in the next sections. The principles are rather general, which means that most of the results are also valid for other air pollution models.

2 Main Physical and Chemical Processes

Five physical and chemical processes have to be described by mathematical terms in the beginning of the development of an air pollution model. These processes are:

- horizontal transport (advection),
- horizontal diffusion,
- chemical transformations in the atmosphere combined with emissions from different sources,
- deposition of pollutants to the surface

and

- vertical exchange (containing both vertical transport and vertical diffusion).

It is important to describe in an adequate way all these processes. However, this is an extremely difficult task; both because of the lack of knowledge for some of the processes (this is mainly true for some chemical reactions and for some of the mechanisms describing the vertical diffusion) and because a very rigorous description of some of the processes will lead to huge computational tasks which may make the treatment of the model practically impossible. The main principles used in the mathematical description of the main physical and chemical processes as well as the need to keep the balance between the rigorous description of the processes and the necessity to be able to run the model on the available computers are discussed in [2] and [5].

3 Mathematical Description of a Large Air Pollution Model

The description of the physical and chemical processes, which are related to transport and transformations of harmful pollutants in the atmosphere, by mathematical terms leads to a system of partial differential equations (PDEs):

$$\begin{aligned}\frac{\partial c_s}{\partial t} = &-\frac{\partial (u c_s)}{\partial x} - \frac{\partial (v c_s)}{\partial y} - \frac{\partial (w c_s)}{\partial z} \\ &+\frac{\partial}{\partial x}\left(K_x \frac{\partial c_s}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_y \frac{\partial c_s}{\partial y}\right) + \frac{\partial}{\partial z}\left(K_z \frac{\partial c_s}{\partial z}\right) \\ &+E_s - (\kappa_{1s} + \kappa_{2s}) c_s + Q_s(c_1, c_2, \dots, c_q), \quad s = 1, 2, \dots, q,\end{aligned} \tag{3.1}$$

where (i) the concentrations of the chemical species are denoted by c_s, (ii) u, v and w are wind velocities, (iii) K_x, K_y and K_z are diffusion coefficients, (iv) the emission sources are described by E_s, (v) κ_{1s} and κ_{2s} are deposition coefficients and (vi) the chemical reactions are denoted by $Q_s(c_1, c_2, \dots, c_q)$.

4 Applying Splitting Procedures

UNI-DEM is split (see [1]) to the following three sub-models:

$$\frac{\partial c_s^{(1)}}{\partial t} = -\frac{\partial (w c_s^{(3)})}{\partial z} + \frac{\partial}{\partial z}\left(K_z \frac{\partial c_s^{(3)}}{\partial z}\right) \tag{4.2}$$

$$\frac{\partial c_s^{(2)}}{\partial t} = -\frac{\partial (u c_s^{(2)})}{\partial x} - \frac{\partial (v c_s^{(2)})}{\partial y} + \frac{\partial}{\partial x}\left(K_x \frac{\partial c_s^{(2)}}{\partial x}\right) + \frac{\partial}{\partial y}\left(K_y \frac{\partial c_s^{(2)}}{\partial y}\right) \tag{4.3}$$

$$\frac{d c_s^{(3)}}{dt} = E_s + Q_s(c_1^{(3)}, c_2^{(3)}, \dots, c_q^{(3)}) - (\kappa_{1s} + \kappa_{2s}) c_s^{(3)} \tag{4.4}$$

The first of these sub-models, (2), describes the vertical exchange. The second sub-model, (3), describes the combined horizontal transport (the advection) and the horizontal diffusion. The last sub-model, (4), describes the chemical reactions together with emission sources and deposition terms. Splitting allows us to apply different numerical methods in the different sub-models and, thus, to reduce considerably the computational work and to exploit better the properties of each sub-model.

5 Numerical Methods

Assume that the space domain is discretized by using a grid with $N_x \times N_y \times N_z$ grid-points, where N_x, N_y and N_z are the numbers of the grid-points along the grid-lines parallel to the Ox, Oy and Oz axes. Assume further that the number of chemical species involved in the model is $q = N_s$. Finally, assume that the spatial derivatives are discretized by some numerical algorithm. Then the systems of PDE's from the previous section will be transformed into systems of ODEs (ordinary differential equations):

$$\frac{dg^{(1)}}{dt} = f^{(1)}(t, g^{(1)}), \quad \frac{dg^{(2)}}{dt} = f^{(2)}(t, g^{(2)}), \quad \frac{dg^{(3)}}{dt} = f^{(3)}(t, g^{(3)}). \tag{5.5}$$

The components of functions $g^{(i)}(t) \in R^{N_x \times N_y \times N_z \times N_s}$, $i = 1, 2, 3$, are the approximations of the concentrations (at time t) at all grid-squares and for all species. The components of functions $f^{(i)}(t, g) \in R^{N_x \times N_y \times N_z \times N_s}$, $i = 1, 2, 3$, depend on the numerical method used in the discretization of the spatial derivatives.

A simple linear finite element method is used to discretize the spatial derivatives in (2) and (3). The spatial derivatives can also be discretized by using other numerical methods as, for example, a pseudospectral discretization, a semi-Lagrangian discretization (can be used only to discretize the first-order derivatives, i.e. the advection part should not be combined with the diffusion part when this method is to be applied) and methods producing non-negative values of the concentrations.

It is necessary to couple the three ODE systems (5). The coupling procedure is connected with the time-integration of these systems. Assume that the values of the concentrations (for all species and at all grid-points) have been found for some $t = t_n$. According to the notation introduced in the previous sub-section, these values can be considered as components of a vector-function $g(t_n) \in R^{N_x \times N_y \times N_z \times N_s}$. The next time-step, time-step $n + 1$ (at which the concentrations are found at $t_{n+1} = t_n + \triangle t$, where $\triangle t$ is some increment), can be performed by integrating successively the three systems. The values of $g(t_n)$ are used as an initial condition in the solution of the first ODE system in (5). The solution of of the first system in (5) is used as an initial condition of the second ODE system in (5). Finally, the solution of the second ODE system in (5) is used as an initial condition of of the third ODE system in (5). The solution of the last ODE system in (5) is used as an approximation to $g(t_{n+1})$. In this way, everything is prepared to start the calculations in the next time-step, step $n + 2$.

The first ODE system in (5), can be solved by using many classical time-integration methods. The so-called θ-method is currently used in UNI-DEM.

Predictor-corrector (PC) methods with several different correctors are used in the solution of the second ODE system in (5). The correctors are carefully chosen so that the stability properties of the method are enhanced. If the code judges the time-stepsize to be too large for the currently used PC method, then it switches to a more stable (but also more expensive, because more corrector formulae are used to obtain stability) PC scheme. On the other hand, if the code judges that the stepsize is too small for the currently used PC method, then it switches to not so stable but more accurate (which is using less corrector formulae and, therefore is less expensive) PC scheme. In this way the code is trying both to keep the same stepsize and to optimize the performance.

The solution of the third system in (5) is much more complicated, because this system is both time-consuming and stiff. Very often the QSSA (Quasi-Steady-State-Approximation) method is used in this part of the model. It is simple and relatively stable but not very accurate (therefore it has to be run with a small time-stepsize). An improved QSSA method was recently implemented in UNI-DEM. The classical numerical methods for stiff ODE systems (such as the Backward Euler Method, the Trapezoidal Rule and Runge-Kutta algorithms) lead to the solution of non-linear systems of algebraic equations and, therefore, they are more expensive. On the other hand, these methods can be incorporated with an error control and perhaps with larger time-steps. Partitioning can also be used. Some convergence problems related to the implementation of partitioning have recently been studied (see [6]).

More details about the numerical methods can be found in [1].

6 Moving from Different Versions to a Common Model

Only two years ago several different versions of the model described by the system of PDEs (1) were available. Six versions were mainly used in the production runs (three 2-D versions discretized on the 96×96, 288×288 and 480×480 grids respectively together with the corresponding three 3-D versions). Recently, these versions were combined in a common model. A special input file, "save_inform" is used to decide how to run the common model. Eight parameters are to be initialized in "save_inform" before the start of the run. These parameters are:

- NX - the number of grid-points along the Ox axis.
- NY - the number of grid-points along the Oy axis.
- NZ - the number of grid-points along the Oz axis.
- $NSPECIES$ - the number of chemical species involved in the model.
- $NREFINED$ - allows us to use refined emissions when available.
- $NSIZE$ - the size of the chunks to be used in the chemical parts.
- $NYEAR$ - the year for which the model is to be run.
- $PATH$ - the working path where the data attached to the different processors will be stored.

There are several restrictions for the parameters, which can be used at present. The restrictions are listed below:

1. The allowed values for NX and NY are 96, 288 and 480. Furthermore, NX must be equal to NY.
2. The allowed values for NZ are 1 (corresponds to the 2-D versions) or 10 (i.e. only 10 layers are allowed for the 3-D versions).
3. Only three chemical schemes (with 35, 56 and 168 species) can be selected at present by setting $NSPECIES$ equal to 35, 56 or 168.
4. Refined emissions are available only for the 288×288 case and will be used when $NREFINED = 1$. If $NREFINED = 0$, then the emissions for the 96×96 grid will be used (simple interpolation will be used when any of the other two grids, the 96×96 grid or the 480×480 grid, is specified).

5. $NSIZE$ must be a divisor of $NX \times NY$ (the value of $NSIZE$ that is used in the runs discussed in section 8 is 48).
6. $NYEAR$ can at present be varied from 1989 to 1998 (we are now working to prolong this interval to year 2003).

Many of these restrictions will be removed (or, at least, relaxed) in the near future. It will, for example, be allowed to (a) specify a rectangular space domain, (b) use more than 10 layers and (c) apply more chemical schemes.

The common model, which can be run as described in this section, is called UNI-DEM (the Unified Danish Eulerian Model).

7 Need for High Performance Computing in the Treatment of Large Air Pollution Models

The computers are becoming more and more powerful. Many tasks, which several years ago had to be handled on powerful supercomputers, can be handled at present on PCs or work-stations. However, there are still many tasks that can only be run on parallel computers. This is especially true for the large air pollution models. The size of the computational tasks in some versions of UNI-DEM is given in the following two paragraphs in order demonstrate the fact that high performance computing is needed when large air pollution models are to be treated.

7.1 Size of the Computational Tasks when 2-D Versions Are Used

Only the last two systems of ODEs in (5) have to be treated in this case. Assume first that the coarse 96×96 grid is used. Then the number of equations in each of the last two systems of ODEs in (5) is equal to the product of the grid points (9216) and the number of chemical species (35), i.e. 322560 equations have to be treated at each time-step when any of the last two systems of ODEs in (5) is handled. The time-stepsize used in the transport sub-model, i.e. the second system of ODEs in (5), is 900 s. This stepsize is too big for the chemical sub-model; the time-stepsize used in the latter model is 150 s. A typical run of this model covers a period of one year (in fact, as mentioned above, very often a period of extra five days is needed to start up the models). This means that 35520 time-steps are needed in the transport sub-model, while six times more time-steps, 213120 time-steps, are needed in the chemical part, the third system of ODEs in (5). If the number of scenarios is not large, then this version of the model can be run on PCs and work-stations. If the number of scenarios is large or if runs over many years have to be performed (which is the case when effects of future climate changes on the air pollution studies is studied), then high performance computations are preferable (this may be the only way to complete the study when either the number of scenarios is very large or the time period is very long).

Assume now that the medium 288×288 grid is used. Since the number of chemical species remains unchanged (35), the number of equations in each of the last two systems in (5) is increased by a factor of 9 (compared with the previous case). This means that 2903040 equations are to be treated at each time step when any of the last two systems of ODEs in (5) is handled. The time-stepsize remains 150 s when the chemical part is treated. The time-stepsize has to be reduced from 900 s to 300 s in the transport part. This means that a typical run (one year + 5 days to start up the model) will require 106760 time-steps when the transport sub-model is treated and 213120 time-steps are needed when the chemical sub-model is handled. Consider the ratio of the computational work when the medium grid is used and the computational work when the coarse grid is used. For the transport sub-model this ratio is 18, while the ratio is 9 for the chemical-sub-model.

Finally, assume that the fine 480×480 grid is used. Using similar arguments as in the previous paragraph, it is easy to show that the number of equations in each of the last two systems of ODEs in (5) is increased by a factor of 25 (compared with the 96x96 grid). This means that 8064000 equations are to be treated at each time step when any of the systems the last two systems of ODEs in (5) is handled. The time-stepsize remains 150 s when the chemical part is treated. The time-stepsize has to be reduced from 900 s to 150 s in the transport part. This means that a typical run (one year + 5 days to start up the model) will require 213520 time-steps for each of the last two systems of ODEs in (5). Consider the ratio of the computational work when the fine grid is used and the computational work when the coarse grid is used. For the transport sub-model this ratio is 150, while the ratio is 25 for the chemical-sub-model. It is clear that this version of the model **must** be treated on powerful parallel architectures.

7.2 Size of the Computational Tasks when 3-D Versions Are Used

All three sub-models, i.e. all three systems of ODEs in (5), have to be treated in this case. Assume that the number of layers in the vertical direction is n ($n = 10$ is used in this paper). Under this assumption the computational work when both the second and the third system of ODEs in (5) are handled by the 3-D versions (either on a coarse grid or on the finer grids) is n times bigger than the computational work for the corresponding 2-D version. The work needed to handle the first of the three systems of ODEs in (5) is extra, but this part of the total computational work is much smaller than the parts needed to treat the second and the third of the three systems of ODEs in (5).

The above analysis of the amount of the computational work shows that it is much more preferable to run the 3-D version on high-speed parallel computers when the coarse grid is used. It will, furthermore, be shown that the runs are very heavy when the 3-D version is to be run on a fine grid. In fact, more powerful parallel computers than the computers available at present are needed if meaningful studies with the 3-D version of UNI-DEM discretized on a fine grid are to be carried out.

8 Parallel Computations

As mentioned in the beginning of the previous section, many tasks, which several years ago had to be handled on powerful supercomputers, can be solved at present on PCs or work-stations. However, it must be emphasized here that the tasks related to the treatment if large-scale are certainly not belonging to the class of tasks that can be handled on PCs or work-stations (see the discussion in the previous section), because several systems of

ODEs containing up to several million equations are to be handled in several hundred thousand time-steps. Moreover, many scenarios are to be run (several hundreds or even several thousands). This is why it is essential

- to exploit efficiently the cache memory of the computers

and

- to develop efficient codes for parallel runs.

Several examples which illustrate the efficiency of UNI-DEM with regard to these two tasks will be given in this section (see more details in [1] and [4]). Computing times achieved by six options run with the chemical scheme with 35 species on 8 processors of a SUN computer are given in Table 1.

Table 1. Total computing times (measured in seconds) which are obtained when six options of UNI-DEM are run on 8 processors

$NX \times NY$	2-D ($NZ = 1$)	3-D ($NZ = 10$)
96	5060	33491
288	70407	549801
480	355387	2559173

It is interesting to see whether increasing the number of processors used by a factor of k will lead to an reduction of the computing time by a factor approximately equal to k. We selected the most time-consuming option (the option discretized on a $480 \times 480 \times 10$ grid) and run it on 32 processors. The results were compared, see Table 2, with the results obtained when 8 processors were used (i.e. we have $k = 4$). It is clearly seen that the speed ups are rather close to linear. It must be stressed however, that the total computing time (about 8.61 days) remains very large also when 32 processors are used.

The conclusion is that more processors and more powerful processors might resolve many of the problems mentioned in the previous sub-section.

Table 2. Comparison of the computing times (measured in seconds) which are obtained on 8 processors and 32 processors with UNI-DEM discretized on a $480 \times 480 \times 10$ grid. The speed-ups obtained in the transition from 8 processors to 32 processors are given in brackets

$Process$	8 processors	32 processors
Hor. adv. + diff.	986672	308035 (3.20)
Chem. + dep.	1055289	268978 (3.92)
Total	2559173	744077 (3.44)

9 Applications of UNI-DEM in Comprehensive Studies

UNI-DEM was run with different options and different scenarios in a long series of comprehensive studies. Several of these studies are listed below (see more details in [2] and [3] as well as in the references there):

- impact of climate changes on air pollution levels,
- influence of the biogenic emissions on the creation of high ozone concentrations,
- long-term variations of the air pollution levels in different parts of Europe,
- influence of European sources outside Denmark on the Danish pollution levels,
- exceedance of critical levels established by EU in Denmark and in other parts of Europe.

10 Conclusions

The main features of a large-scale air pollution model, the Unified Danish Eulerian Model (UNI-DEM), are described in this paper. The discretization of this model over an area containing the whole of Europe leads to the necessity to solve systems containing many millions of equations at every time-step (the number of time-steps being normally several hundred thousand). This is why it is only possible to carry out successfully long series of runs (with many scenarios) when modern high-speed computers are available and efficiently used. The code is performing very well on parallel computers. This fact allowed us to perform many important real-life studies with the model.

Acknowledgements

A grant (CPU-1101-17) from the Danish Centre for Scientific Computing (DCSC) gave us access to the Sun computers at the Technical University of Denmark. The members of the staff of DCSC helped us to resolve some difficult problems related to the efficient exploitation of the grid of Sun computers.

The work was partly supported by a grant from NATO (Collaborative Linkage Grant EST.CLG.980505) in which scientists from five countries united their efforts in a study on the *"Impact of Climate Changes on Pollution Levels in Europe"*.

References

1. Alexandrov, V., Owcharz, W., Thomsen, P. G. and Zlatev, Z.: Parallel runs of a large air pollution model on a grid of Sun computers. Mathematics and Computers in Simulation, Vol. 65 (2004), 557-577.
2. Dimov, I. and Zlatev, Z.: Computational and Numerical Challenges in Air Pollution Modelling. Frontiers in Applied Mathematics, SIAM, Philadelphia, to appear.
3. Havasi, A. and Zlatev, Z.: Trends of Hungarian air pollution levels on a long time-scale. Atmospheric Environment, Vol 36 (2002), 4145-4156.
4. Owczarz, W. and Zlatev, Z.: Parallel matrix computations in air pollution modelling. Parallel Computing, Vol. 28 (2002), 355-368.

5. Zlatev, Z.: Computer treatment of large air pollution models,. Kluwer Academic Publishers, Dordrecht-Boston-London, 1995.
6. Zlatev, Z.: Partitioning ODE systems with an application to air pollution models. Computers and Mathematics with Applications, Vol. 42 (2001), 817-832.
7. Zlatev, Z.: Massive data set issues in air pollution modelling. In: Handbook on Massive Data Sets (J. Abello, P. M. Pardalos and M. G. C. Resende, eds.), pp. 1169-1220. Kluwer Academic Publishers, Dordrecht-Boston-London, 2002.

Interval Methods: An Introduction

Organizers: Luke E.K. Achenie[1], Vladik Kreinovich[2], and Kaj Madsen[3]

[1] Department of Chemical Engineering, Unit 3222
University of Connecticut, Storrs, CT, USA
achenie@engr.uconn.edu

[2] Department of Computer Science, University of Texas
El Paso, TX 79968, USA
vladik@cs.utep.edu

[3] Department of Informatics and Mathematical Modelling, Technical University of Denmark
DK-2800 Lyngby, Denmark
km@imm.dtu.dk

Abstract. This chapter contains selected papers presented at the Minisymposium on Interval Methods of the PARA'04 Workshop "State-of-the-Art in Scientific Computing". The emphasis of the workshop was on high-performance computing (HPC). The ongoing development of ever more advanced computers provides the potential for solving increasingly difficult computational problems. However, given the complexity of modern computer architectures, the task of realizing this potential needs careful attention. A main concern of HPC is the development of software that optimizes the performance of a given computer.
An important characteristic of the computer performance in scientific computing is the accuracy of the computation results. Often, we can estimate this accuracy by using traditional statistical techniques. However, in many practical situations, we do not know the probability distributions of different measurement, estimation, and/or roundoff errors, we only know estimates of the upper bounds on the corresponding measurement errors, i.e., we only know an *interval* of possible values of each such error. The papers from the following chapter contain the description of the corresponding "interval computation" techniques, and the applications of these techniques to various problems of scientific computing.

Why Data Processing?

In many real-life situations, we are interested in the value of a physical quantity y that is difficult or impossible to measure directly. Examples of such quantities are the distance to a star and the amount of oil in a given well. Since we cannot measure y directly, a natural idea is to measure y *indirectly*. Specifically, we find some easier-to-measure quantities $x_1, \ldots, x_n$ which are related to y by a known relation $y = f(x_1, \ldots, x_n)$; this relation may be a simple functional transformation, or complex algorithm (e.g., for the amount of oil, numerical solution to an inverse problem). Then, to estimate y, we first measure the values of the quantities $x_1, \ldots, x_n$, and then we use the results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of these measurements to to compute an estimate $\widetilde{y}$ for y as $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$.

For example, to find the resistance R, we measure current I and voltage V, and then use the known relation $R = V/I$ to estimate resistance as $\widetilde{R} = \widetilde{V}/\widetilde{I}$.

In this example, the relation between x_i and y is known exactly; in many practical situations, we only known an approximate relation $y \approx \widetilde{f}(x_1, \ldots, x_n)$ between x_i and y. In such situations, the estimate $\widetilde{y}$ for y is computed as $\widetilde{y} = \widetilde{f}(\widetilde{x}_1, \ldots, \widetilde{x}_n)$.

Computing an estimate for y based on the results of direct measurements is called *data processing*; data processing is the main reason why computers were invented in the first place, and data processing is still one of the main uses of computers as number crunching devices.

Why Interval Computations? From Computing to Probabilities to Intervals

Measurement are never 100% accurate, so in reality, the actual value x_i of i-th measured quantity can differ from the measurement result $\widetilde{x}_i$. Because of these *measurement errors* $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$, the result $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ of data processing is, in general, different from the actual value $y = f(x_1, \ldots, x_n)$ of the desired quantity y [6].

It is desirable to describe the error $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$ of the result of data processing. To do that, we must have some information about the errors of direct measurements.

What do we know about the errors Δx_i of direct measurements? First, the manufacturer of the measuring instrument must supply us with an estimate of the upper bound Δ_i on the measurement error. (If no such upper bound is supplied, this means that no accuracy is guaranteed, and the corresponding "measuring instrument" is practically useless.) In this case, once we performed a measurement and got a measurement result $\widetilde{x}_i$, we know that the actual (unknown) value x_i of the measured quantity belongs to the interval $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$, where $\underline{x}_i = \widetilde{x}_i - \Delta_i$ and $\overline{x}_i = \widetilde{x}_i + \Delta_i$.

In many practical situations, we not only know the interval $[-\Delta_i, \Delta_i]$ of possible values of the measurement error; we also know the probability of different values Δx_i within this interval. This knowledge underlies the traditional engineering approach to estimating the error of indirect measurement, in which we assume that we know the probability distributions for measurement errors Δx_i.

In practice, we can determine the desired probabilities of different values of Δx_i by comparing the results of measuring with this instrument with the results of measuring the same quantity by a standard (much more accurate) measuring instrument. Since the standard measuring instrument is much more accurate than the current one, the difference between these two measurement results is practically equal to the measurement error; thus, the empirical distribution of this difference is close to the desired probability distribution for measurement error. There are two cases, however, when this determination is not done:

– First is the case of cutting-edge measurements, e.g., measurements in fundamental science. When a Hubble telescope detects the light from a distant galaxy, there is no "standard" (much more accurate) telescope floating nearby that we can use to calibrate the Hubble: the Hubble telescope is the best we have.
– The second case is the case of measurements on the shop floor. In this case, in principle, every sensor can be thoroughly calibrated, but sensor calibration is so costly – usually costing ten times more than the sensor itself – that manufacturers rarely do it.

In both cases, we have no information about the probabilities of Δx_i; the only information we have is an estimate on the upper bound on the measurement error.

In this case, after we performed a measurement and got a measurement result $\widetilde{x}_i$, the only information that we have about the actual value x_i of the measured quantity is that it belongs to the interval $\mathbf{x}_i = [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$. In such situations, the only information that we have about the (unknown) actual value of $y = f(x_1, \dots, x_n)$ is that y belongs to the range $\mathbf{y} = [\underline{y}, \overline{y}]$ of the function f over the box $\mathbf{x}_1 \times \dots \times \mathbf{x}_n$:

$$\mathbf{y} = [\underline{y}, \overline{y}] = \{f(x_1, \dots, x_n) \,|\, x_1 \in \mathbf{x}_1, \dots, x_n \in \mathbf{x}_n\}.$$

The process of computing this interval range based on the input intervals $\mathbf{x}_i$ is called *interval computations*; see, e.g., [1,2,3,5].

Comment. In addition to measurement errors, we also have round-off errors and – in case some parameters are estimated by experts – also uncertainty of expert estimates.

Interval Computations Techniques: Brief Reminder

Historically the first method for computing the enclosure for the range is the method which is sometimes called “straightforward" interval computations. This method is based on the fact that inside the computer, every algorithm consists of elementary operations (arithmetic operations, $\min$, $\max$, etc.). For each elementary operation $f(a, b)$, if we know the intervals $\mathbf{a}$ and $\mathbf{b}$ for a and b, we can compute the exact range $f(\mathbf{a}, \mathbf{b})$. The corresponding formulas form the so-called *interval arithmetic*. For example,

$$[\underline{a}, \overline{a}] + [\underline{b}, \overline{b}] = [\underline{a} + \underline{b}, \overline{a} + \overline{b}]; \;\; [\underline{a}, \overline{a}] - [\underline{b}, \overline{b}] = [\underline{a} - \overline{b}, \overline{a} - \underline{b}];$$

$$[\underline{a}, \overline{a}] \cdot [\underline{b}, \overline{b}] = [\min(\underline{a} \cdot \underline{b}, \underline{a} \cdot \overline{b}, \overline{a} \cdot \underline{b}, \overline{a} \cdot \overline{b}), \max(\underline{a} \cdot \underline{b}, \underline{a} \cdot \overline{b}, \overline{a} \cdot \underline{b}, \overline{a} \cdot \overline{b})].$$

In straightforward interval computations, we repeat the computations forming the program f step-by-step, replacing each operation with real numbers by the corresponding operation of interval arithmetic. It is known that, as a result, we get an enclosure $\mathbf{Y} \supseteq \mathbf{y}$ for the desired range.

In some cases, this enclosure is exact. In more complex cases (see examples below), the enclosure has excess width.

There exist more sophisticated techniques for producing a narrower enclosure, e.g., a centered form method. However, for each of these techniques, there are cases when we get an excess width. Reason: as shown in [4,7], the problem of computing the exact range is known to be NP-hard even for polynomial functions $f(x_1, \dots, x_n)$ (actually, even for quadratic functions f).

Applications of Interval Techniques

The ultimate objective of interval computations has always been to apply these methods to practical problems. The workshop provided us with a unique opportunity to promote collaboration between interval researchers and researcher interested in applications.

This chapter contains extended versions of selected papers presented at the interval Minisymposium of PARA’04.

Acknowledgments

First of all, we would like to thank the organizers of PARA'04, especially Jerzy Wasniewski and Kaj Madsen (Technical University of Denmark) and Jack Dongarra (University of Tennessee and Oak Ridge National Laboratory), for their wonderful organization, their enormous support and help. We also want to thank the anonymous referees for their help in reviewing the papers, all the attendees of the minisymposium for their fruitful discussions, and, of course, we want to thank all the authors of the papers, for their hard work and exciting results.

References

1. Jaulin L., Keiffer M., Didrit O., and Walter E. (2001), "Applied Interval Analysis", Springer-Verlag, Berlin.
2. Kearfott R. B. (1996), "Rigorous Global Search: Continuous Problems", Kluwer, Dordrecht.
3. Kearfott R. B. and Kreinovich V., eds. (1996), "Applications of Interval Computations" (Pardalos. P. M., Hearn, D., "Applied Optimization", Vol. 3), Kluwer, Dordrecht.
4. Kreinovich V., Lakeyev A., Rohn J., and Kahl P. (1997), "Computational Complexity and Feasibility of Data Processing and Interval Computations" (Pardalos. P. M., Hearn, D., "Applied Optimization", Vol. 10), Kluwer, Dordrecht.
5. Moore R. E. (1979), "Methods and Applications of Interval Analysis", SIAM, Philadelphia.
6. Rabinovich S. (1993), "Measurement Errors: Theory and Practice", American Institute of Physics, New York.
7. Vavasis S. A. (1991), "Nonlinear Optimization: Complexity Issues", Oxford University Press, N.Y.

A Chemical Engineering Challenge Problem That Can Benefit from Interval Methods

Luke E.K. Achenie and Gennadi M. Ostrovsky

Department of Chemical Engineering, Unit 3222
University of Connecticut, Storrs, CT, USA

Abstract. Computational mathematics has been an integral part of Chemical Engineering since the early sixties. Unfortunately, researchers within the Applied and Computational Mathematics community are only minimally aware of this fact. Here we will discuss one challenge problem in Chemical Engineering which requires computational mathematics tools and may benefit greatly from interval analysis. We will also briefly introduce a second challenge problem.

1 Introduction

Computational mathematics has been an integral part of Chemical Engineering since the early sixties. Unfortunately, researchers within the Applied and Computational Mathematics community are only minimally aware of this fact. The goal of this paper is to discuss a challenging, unsolved or partially solved problem in Chemical Engineering which requires computational mathematics tools. An example challenge problem comes from process simulation that involves hundreds and sometimes thousands of process variables and/or highly coupled nonlinear process constraints. One of the issues is propagating uncertainties in the process model parameters throughout the flowsheet. Specifically one needs to address the question: given a set of uncertain parameters with known bounds, can all the process constraints be satisfied for all realizations of the uncertain parameters? How large can the domain of an uncertain parameter be without violating any process constraint? What is the limiting process constraint? These considerations lead to what is termed "flexibility analysis" within the Chemical Engineering literature. Flexibility analysis sub-problems have been modeled as *maxminmax* nonlinear programs. These problems are inherently non-linear, non-convex and non-smooth. Can interval methods be used to solve flexibility analysis sub-problems? It is the authors' view that this may very well be the case.

2 Challenge Problem 1 – Flexibility Analysis

The problem of optimal design using mathematical models with bounded uncertain parameters θ is very important in chemical engineering. We refer to this problem as the *flexibility analysis problem*. A formulation of the problem is given in the form of the two-step optimization problem (TSOP) [1].

$$f = \min_{d,z^i} \sum_{i \in I_1} \omega_i f(d, z^i, \theta^i) \tag{2.1}$$

$$g_j(d, z^i, \theta^i) \le 0 \;\; j \in J \;\; i \in I_1 \;\; \theta \in T$$

$$\chi(d) \le 0 \tag{2.2}$$

where the feasibility function $\chi(d)$ is expressed as

$$\chi(d) = \max_{\theta \in T} \min_{z \in Z} \max_{j \in J} g_j(d, z, \theta) \;\; J = (1, \ldots, m) \tag{2.3}$$

The *maxminmax* operator in (2.3) introduces non-differentiability and multiextremality in the optimization problem. To address the non-differentiability issue, we have developed a *split and bound* (SB) strategy, which systematically provides lower and upper bounds to the objective function in (2.1) in the space of the constraints. The lower and upper bound problems are standard NLP problems. To show the suggested approach, we proceed as follows.

The feasibility function can be represented as

$$\chi(d) = \max_{\theta \in T} \psi(d, \theta) \tag{2.4}$$

where

$$\psi(d, \theta) = \min_{z \in Z} \max_{j \in J} g_j(d, z, \theta) \tag{2.5}$$

The following relations hold

$$\max_x \max_y f(x, y) = \max_y \max_x f(x, y) \tag{2.6}$$

$$\min_x \max_y f(x, y) \ge \max_y \min_x f(x, y) \tag{2.7}$$

$$\max \varphi(x) \le 0 \leftrightarrow \varphi(x) \le 0, \forall x \in X \tag{2.8}$$

where x and y are vectors of continuous or discrete variables. The relations in (2.6) and (2.8) are fairly straightforward, while (2.7) is proved in [4].

Theorem 1: (see [5]) *If* $[x^*, y^{i*}]$ *is the solution of*

$$\overline{f} = \min_{x, y^i} f(x)$$

$$\varphi_i(x, y^i) \le 0 \;\; i = 1, \ldots, m$$

then x^* *is the solution of*

$$f = \min_{x \in X} f(x) \tag{2.9}$$

$$\min_{y \in Y} \varphi_i(x, y) \le 0 \;\; i = 1, \ldots, m$$

Let T be partitioned into N subregions T_i: $T = T_1 \cup T_2 \cup \ldots \cup T_N$. For each subregion T_i define

$$\chi^{U,i}(d) = \min_z \max_{\theta \in T_i} \max_{j \in J} g_j(d, z, \theta) \tag{2.10}$$

This function is obtained by rearrangement of the first two operations in (2.3). If T_i coincides with T then we will designate the function as $\chi^U(d)$. Using inequality (2.7) one can show that $\chi^U(d)$ is an upper bound of $\chi(d)$ on T

$$\chi^U(d) \geq \chi(d) \equiv \max_{\theta \in T} \psi(d, \theta)$$

Similarly for $\chi^{U,i}$ we have

$$\chi^{U,i}(d) \geq \max_{\theta \in T_i} \psi(d, \theta) \tag{2.11}$$

Then from (2.11)

$$\max_i \chi^{U,i}(d) \geq \max_i \max_{\theta \in T_i} \psi(d, \theta) = \max_{\theta \in T} \psi(d, \theta) = \chi \tag{2.12}$$

2.1 A Point Arithmetic Strategy for the Two-Step Optimization Problem

We consider a point arithmetic strategy (namely, the split and bound (SB) algorithm) for solving the TSOP. The SB is a two-level iterative procedure, which employs a partitioning of T into subregions $T_i^{(k)}$. It has the following two main operations: (1) An algorithm for estimating an upper bound $f^{U,(k)}$ of the TSOP objective function (2) An algorithm for estimating a lower bound $f^{L,(k)}$ of the TSOP objective function. The upper level serves to partition T using information obtained from the lower level. The lower level is used for calculation of lower and upper bounds.

Consider the algorithm for calculation of an upper bound of the TSOP objective function. At the *k-th* iteration in the SB method, let T be partitioned into the subregions $T_i^{(k)}$, $i = 1, \ldots, N_k$. On $T_i^{(k)}$ an upper bound of the TSOP is given by

$$f^{U,(k)} = \min_{d, z^i} \sum_{i \in I_1} \omega_i f(d, z^i, \theta^i) \tag{2.13}$$

$$g_j(d, z^i, \theta^i) \leq 0 \quad j \in J \quad i \in I_1 \quad J = (1, \ldots, n)$$

$$\chi^{U,1}(d) \leq 0, \ldots, \chi^{U,N_k}(d) \leq 0 \tag{2.14}$$

where $\chi^{U,i}$ is determined by (2.10). Using $\chi^{U,i}$ from (2.10), Theorem 1 and relation (2.6) we can transform (2.13) and (2.14) as

$$f^{U,(k)} = \min_{d, z^i} \sum_{i \in I_1} \omega_i f(d, z^i, \theta^i) \tag{2.15}$$

$$g_j(d, z^i, \theta^i) \leq 0 \quad j \in J \quad i \in I_1$$

$$\max_{\theta \in T_i} g_j(d, z^i, \theta) \leq 0 \quad l = 1, \ldots, N_k, j = 1, \ldots, m \tag{2.16}$$

Let $[d^{(k)}, z^{i,(k)}, z^{l,(k)}]$ be the solution of the above problem. Since condition (2.12) is met, the feasible region of (2.13) is contained in that of the TSOP (see (2.1)). Consequently $f^{U,(k)}$ is an upper bound of the TSOP objective function, thus $f^{U,(k)} \geq f$.

Problem (2.15) is a semi-infinite programming problem for which the outer approximation algorithm [2] can be used.

Consider the lower bound problem for the TSOP. Using relation (2.6) and $\chi_2(d)$ in the form (2.4), we can reduce the TSOP to

$$f = \min_{d,z^i} \sum_{i\in I_1} \omega_i f(d, z^i, \theta^i) \tag{2.17}$$

$$g_j(d, z^i, \theta^i) \le 0 \;\; j \in J \;\; i \in I_1$$

$$\psi(d, \theta^i) \le 0 \qquad \forall \theta^i \in T \tag{2.18}$$

This is an optimization problem with an infinite number of constraints (2.18). Let $[d^*, z^{i*}]$ be the solution of the problem. At the *k-th* iteration, define the set $S_2^{(k)} = \{\theta^{1,r} : r \in I_2^{(k)}\}$, where $I_2^{(k)}$ is a set of indices of the points belonging to $S_2^{(k)}$. We refer to these points as *critical points*. Consider the following problem:

$$f^{L,(k)} = \min_{d,z^i} \sum_{i\in I_1} \omega_i f(d, z^i, \theta^i)$$

$$g_j(d, z^i, \theta^i) \le 0 \;\; j \in J \;\; i \in I_1$$

$$\psi(d, \theta_i) \le 0 \;\; \forall \theta^i \in S_2^{(k)} \tag{2.19}$$

In contrast to (2.17), this problem has a finite number of constraints. Taking into account the form of $\psi(d, \theta)$ (see (2.5)) and using Theorem 1 and relation (2.6) we can transform problem (2.19) as

$$f^{L,(k)} = \min_{d,z^i} \sum_{i\in I_1} \omega_i f(d, z^i, \theta^i) \tag{2.20}$$

$$g_j(d, z^i, \theta^i) \le 0 \;\; j \in J \;\; i \in I_1$$

$$g_j(d, z^{1,r}, \theta^{1,r}, \theta^2) \le 0 \;\; \forall \theta^r \in S_2^{(k)} \;\; j = 1, \ldots, m$$

Since $S_2^{(k)} \in T$ then the feasible region of problem (2.17) will be contained in the feasible region of (2.19), therefore $f^{L,(k)}$ is a lower bound on the TSOP objective function, thus $f^{L,(k)} \le f$.

In the proposed lower bound algorithm, we need rules for partitioning and selection of the set $S_2^{(k)}$of critical points. For partitioning we will use the following heuristic: at the k-th iteration, only those subregions $T_l^{(k)}$ $(l = 1, \ldots, N_k)$ for which constraints (2.16) (in the upper bound problem) are active are partitioned as follows

$$\exists j \in J \max_{\theta\in T_l^1} g_j(d^{(k)}, z^{l,(k)}, \theta) = 0.$$

The rationale for the above heuristic is as follows. Partitioning the subregions with active constraints in (2.15) removes the constraints from the problem. It is well established that deletion of active constraints leads to an optimal objective function value which is at least as good as before. In most cases it leads to an improvement in the optimal value of the objective function.

Now consider the formation of the set of critical point $S_2^{(k)}$. Let $\theta^{l,j}$ be the solution of the problem in the left-hand side of (2.16) which are active then the set $S_2^{(k)}$ will have the following form: $S_2^{(k)} = \{\theta^{l,j}, l = 1, \dots, N_k,\ j = 1, \dots, m\}$. We will refer to these points as *active points*.

Introduce a set $L^{(k)}$ of subregions $T_i^{(k)}$ as follows: $L^{(k)} = \{T_i^{(k)} : r(T_i^{(k)}) > \delta\}$, where δ is a small positive scalar. Now we propose the following algorithm for solving the TSOP.

Algorithm 1

Step 1: Set k=1. Initialize the set of approximation points, $d^{(0)}, z^{i,(0)}, z^{l,(0)}$ $(l = 1, \dots, N_1)$ and δ.

Step 2: Solve the upper bound problem to obtain the solution

$$[d^{(k)}, z^{i,(k)}, z^{l,(k)}] \ (i \in I_1,\ \ l = 1, \dots, N_k)$$

Step 3: Determine the set $Q^{(k)}$ of subregions with active constraints in problem (2.13) as

$$\chi^{U,i}(d^{(k)}) = 0,\ \ T_i^{(k)} \in Q^{(k)}$$

Step 4: Calculate a lower bound by solving problem (2.20).

Step 5: If

$$f_2^{U,(k)} - f_2^{L,(k)} \le \varepsilon f_2^{U,(k)} \tag{2.21}$$

is met then the solution of TSOP is found, therefore stop.

Step 6: If

$$r(T_i^{(k)}) \le \delta,\ \ i = 1, \dots, N_k$$

is met go to Step 8.

Step 7: Find the set $R^{(k)} = L^{(k)} \cap Q^{(k)}$. Partition each $T_i^{(k)} \in R^{(k)}$ into two subregions.

Step 8: Set $\delta = \delta/2$ and go to Step 7.

Step 9: Set $k = k + 1$ and go to Step 2.

The SB method will obtain at least a local minimum of TSOP2. Determining an initial partitioning of T into subregions $T_i^{(1)}$ $(i = 1, \dots, N_1)$, is not simple. Of course, one can select an initial partition which consists of one subregion, namely the whole of T. However, in this case it is quite possible that the upper bound problem (2.15) has no solution. Therefore we need a strategy for determination of the initial partition. For this we propose the following

$$\min_{\substack{d,u \\ \chi(d) \le u}} \ u \tag{2.22}$$

Let d^* be the solution of the problem. It is clear, that if $\chi_2(d^*) \ge 0$ then TSOP has no solution. On the other hand if $\chi_2(d^*) \le 0$ then we obtain the design d^* and a partition for which there is a solution to the upper bound problem (2.15). It is easy to see that (2.22) is a particular case of the TSOP. Therefore we can use Algorithm 1.

2.2 Computational Experiments

Using the algorithms above, we optimized (under parametric uncertainty) a chemical process consisting of a reactor and heat exchanger on a 2GHz Pentium 4 PC. Formulation of the problem is in [1]. The uncertainty region is represented as: $T(\gamma) = \{\theta_i : \theta_i^N - \gamma\Delta\theta_i \leq \theta_i \leq \theta_i^N + \gamma\Delta\theta_i\}$ where θ_i^N is the nominal value of θ_i, $\delta\theta_i$ is the maximal deviation from the nominal values and γ is a positive scalar. We employed the Knitro software [6] for all the NLP subproblems in Algorithm 1.

Nominal optimization (i.e. no parametric uncertainty) is compared with the TSOP (i.e. with parametric uncertainty) in Table 1. In the first row we give the results of nominal optimization ($\gamma = 0$). In the first column, *nonlinear (linear)* means that in the maximization in (2.23) of Algorithm 1, nonlinear (linear) models are used. In columns 3, 4, and 5, the optimal value of the objective function, the volume and the heat exchanger area are given. In columns 6 and 7, we give the CPU time (t_F) for solving problem of determination of an initial partition, and the CPU time (t_1) for TSOP1.

Table 1. Nominal optimization and TSOP

Max alg.	γ	F \$/yr	V m^3	A m^2	t_F sec	t_1 sec
–	0.0	9769.25	5.42	7.30	–	–
nonlinear	1.0	11045.79	6.63	9.28	0	2.19
linear	1.0	11045.79	6.63	9.28	0	0.36
nonlinear	1.5	12078.15	7.47	11.05	7.26	124.14
linear	1.5	12078.15	7.47	11.05	0.72	3.83
nonlinear	1.75	12685.86	8.01	12.18	21.55	427.41
linear	1.75	12685.86	8.01	12.18	1.83	10.80

2.3 What Role Can Interval Analysis Play in the Solution of the Two-Step Optimization Problem?

From Eqn. (2.3) the feasibility function $\chi(d)$ is expressed as

$$\chi(d) = \max_{\theta\in T} \min_{z\in Z} \max_{j\in J} g_j(d,z,\theta) \quad J = (1,\ldots,m) \tag{2.23}$$

Then the TSOP problem can be expressed as

$$f = \min_{\substack{d,z^i \\ g_j(d,z^i,\theta^i)\leq 0 \\ \chi(d)\leq 0}} \sum_{i\in I_1} \omega_i f(d,z^i,\theta^i) \quad j\in J \quad i\in I_1 \quad \theta\in T \tag{2.24}$$

Consider all the variables $[d,\theta,z]$ to be interval variables and the parameters $[z^i,\theta^i,\omega_i]$ to be point valued. Next designate $[d,\theta,z]$ as $[p_3,p_2,p_1]$, then as shown in [3], Eqn. (2.23) is equivalent to

$$\chi(d) = \max_{p_2} \min_{\substack{p_1 \\ g_j(d,z,\theta)\leq u \quad J=(1,\ldots,m)}} u \tag{2.25}$$

Thus the TSOP problem is of the form

$$f = \min_{p_3} \max_{p_2} \min_{\substack{p_1 \\ g_j(d,z,\theta) \leq u \quad J=(1,\ldots,m)}} u \tag{2.26}$$

In [3], an algorithm has been described for the guaranteed solution of problem (2.26) and has largely been demonstrated on unconstrained problem. In contrast the TSOP problem when cast in the form of (2.26) has several nonconvex and possibly nonsmooth constraints. Thus the algorithm in [3] needs to be modified/tailored to the TSOP problem. Assuming one is able to do this, then the interval arithmetic approach can have at least one important advantage over the point arithmetic approach described in Section 2 as follows.

- In Section 2, Algorithm 1 for solving the TSOP problem relies on certain convexity assumptions. In the absence of these assumptions, global optimization (which is computationally intensive) will have to be employed in the sub-problems described under Algorithm 1. In contrast the interval approach does not make any convexity assumptions.
- The ability to handle problems with all types of convexity characteristics and non-smoothness characteristics increases the class of TSOP problems that can be solved.

The jury is still out with regard to whether the point arithmetic approach is computationally more efficient than the interval approach. However, it is most likely the case that the curse of dimensionality will have a more profound effect on the interval approach. In addition we expect that the *dependency effect* which is possibly the Achilles' heel of interval computations may play a big role especially in the presence of the kind of process constraints we encounter.

3 Challenge Problem 2 – Blackbox Process Models

In keeping with the theme of the paper as suggested in the title, this section makes a passing remark at another problem in Chemical Engineering that can potentially benefit from interval analysis. In chemical process simulation, we tend to deal with blackbox models such as those from a simulation engine. The main types of blackbox models are as follows: (1) can be expressed analytically but the user is not allowed direct access to the model (proprietary software), (2) can be expressed only algorithmically but the user is not allowed direct access to the model, (3) can be expressed only algorithmically and the user is allowed direct access to the model. As one reviewer pointed out, interval methods had and could be used to estimate not only the range of a function given by an analytical formula, but also the range of a function given by a more general algorithm. This would be true only for case (3). Cases (1) and (2) are quite common in Chemical Engineering practice; therefore development of interval approaches that can handle cases (1) and (2) will be very helpful.

4 Conclusions

Two examples of challenge problems in chemical engineering have been presented. The first one presented in greater detail involves propagating uncertainties in the process model parameters throughout the flowsheet. Specifically one needs to address the question: given a set of uncertain parameters with known bounds, can all the process constraints be satisfied for all realizations of the uncertain parameters? How large can the domain of an uncertain parameter be without violating any process constraint? What is the limiting process constraint? These considerations lead to what is termed \flexibility analysis" within the Chemical Engineering literature. Flexibility analysis sub-problems have been modeled as *maxminmax* nonlinear programs. These problems are inherently non-linear, non-convex and non-smooth. Here in this paper a point arithmetic based approach and a possible interval approach have been outlined for solving one aspect (namely the TSOP) of the flexibility problem. The second challenge problem deals with blackbox models. The broad concepts for using interval analysis to handle the two challenge problems presented here, are being explored by our group. Additionally, we would like to encourage the theory experts in the interval community to bring their craft to bear on problems of importance to the Chemical Engineering community.

References

1. K.P. Halemane and I.E. Grossmann, "Optimal process design under uncertainty," vol. 29, pp. 425–433, 1983.
2. R. Hettich and K.O. Kortanek, "Semi-infinite programming: Theory, methods and applications," vol. 35(3), pp. 380–429, 1993.
3. L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis*. London, UK: Springer, 2001.
4. J.C. McKinsey, *Introduction to the Theory of Games*. New York, NY: McGraw-Hill, 1952.
5. G.M. Ostrovsky, Y.M. Volin, and D. Golovashkin, "Evaluation of chemical process flexibility," vol. 20, pp. S617–S622, 1996.
6. R.A. Waltz and J. Nocedal, *Knitro 2.0 User's Handbook*, Northwestern University, 2002.

Performance of Taylor Model Methods for Validated Integration of ODEs

Martin Berz and Kyoko Makino

Department of Physics and Astronomy
Michigan State University
East Lansing, MI 48824, USA

Abstract. The performance of various Taylor model (TM)-based methods for the validated integration of ODEs is studied for some representative computational problems. For nonlinear problems, the advantage of the method lies in the ability to retain dependencies of final conditions on initial conditions to high order, leading to the ability to treat large boxes of initial conditions for extended periods of time. For linear problems, the asymptotic behavior of the error of the methods is seen to be similar to that of non-validated integrators.

1 Introduction

Taylor model methods provide functional inclusions of a function f of the form

$$f(x) \in P(x) + I \text{ for } x \in B$$

where B is the domain box, P is the nth order Taylor polynomial of f around a point $x_0 \in B \subset R^m$ expressed by floating point coefficients, and I is an interval remainder bound including errors associated to the floating point representation of P. Taylor models of a given function can be obtained from its code list by use of Taylor model arithmetic. Compared to other validated methods, the approach has the following advantages:

- Reduction of the dependency problem [1], since the bulk of the functional dependency is represented by the polynomial P
- High-order approximation, since the width of I scales with order $(n + 1)$ if n is the order of P [2]
- Simplification of global optimization, since original function is replaced by P, which has little dependency and is easily amenable to domain reduction
- Availability of advanced tools for solutions of implicit equations

The methods also allow the development of validated integration schemes [3] [4] that represent the dependence of the solution at the time t_k after the kth step in terms of the initial conditions x_i as a Taylor model

$$x(x_i, t_k) \in P_k(x_i) + I_k.$$

Frequently the dynamics is also represented in a pre-conditioned form by factoring a co-moving coordinate frame as

$$x(x_i, t_k) \in (C_k + J_k) \circ (P_{rk}(x_i) + I_{rk})$$

where C_k represents a change of variables. Frequently used choices in the study of dynamics are linear transformations based on curvilinear coordinates (PC-CV) [5] [4] [6] [7] [8] and blunted parallelepiped coordinates (PC-BL), which prevents the occurring parallelepipeds from becoming ill-conditioned [9]. For purposes of comparison, we also utilize the QR coordinate system used for the description of the error term by Lohner in his code AWA [10] [11] [12], which determines a linear relationship between final conditions and initial conditions.

Finally it is also possible to give up the direct connection between final conditions and initial conditions and merely ask that

$$x\left(x_i, t_k\right) \in \bigcup_{x_i \in B} P_k^*\left(x_i, t_k\right)$$

where P_k^* is a polynomial obtained in the so-called shrink wrap approach [13] [9], of which we use the blunted version (SW-BL). All these features are implemented in the code COSY-VI.

2 Nonlinear Problems

As an example for the behavior for a nonlinear problem, we utilize a classical example from the literature [14] [15] of validated integration of ODEs, the Volterra equations

$$\frac{dx_1}{dt} = 2x_1(1 - x_2), \quad \frac{dx_2}{dt} = -x_2(1 - x_1). \tag{2.1}$$

The ODEs admit an invariant which has the form

$$C(x_1, x_2) = x_1 x_2^2 e^{-x_1 - 2x_2} = \text{Constant}, \tag{2.2}$$

which is useful for the practical study of the methods under consideration. In the quadrant characterized by $x_{1,2} > 0$, the constant is positive, which implies that contour lines are restricted to this quadrant and even form closed curves. Figure 1 illustrates the shape of C and a few of its contour lines. The period of one cycle of the solution depends on the initial condition, and outer orbits take longer.

We study the ODEs for the initial conditions

$$x_{01} \in 1 + [-0.05, 0.05], \quad x_{02} \in 3 + [-0.05, 0.05].$$

In the contour line plot of the invariant in figure 1, the center of these initial conditions lies on the outermost of the three shown contour lines. Within the Taylor model framework of COSY-VI, the initial conditions are represented by the initial Taylor models

```
I  COEFFICIENT              ORDER EXPONENTS
1   1.000000000000000         0    0 0  0
2  0.5000000000000000E-01     1    1 0  0
R  [-.5884182030513415E-015,0.5884182030513415E-015]

1   3.000000000000000         0    0 0  0
2  0.5000000000000000E-01     1    0 1  0
R  [-.1476596622751470E-014,0.1476596622751470E-014]
```

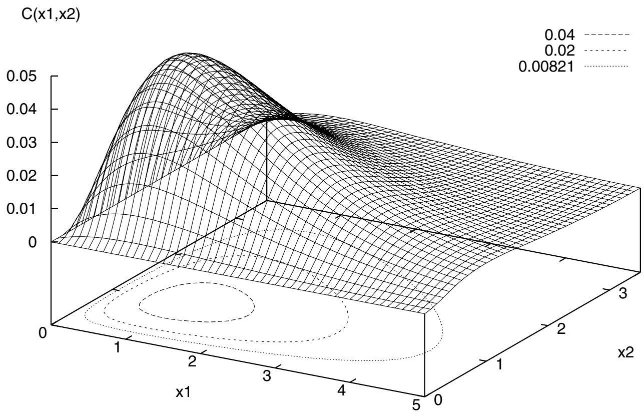


Fig. 1. The invariant of the Volterra equations, which has the form $C(x_1, x_2) = x_1 x_2^2 e^{-x_1-2x_2}$. A few contour lines are shown, including the one passing through the initial condition $(x_1, x_2) = (1, 3)$ being studied

We use the ODEs to compare the behavior of the code COSY-VI and the code AWA by Lohner [12] and to study the longer term behavior of the validated integration process. As a first step, in figure 2 we show the overall width of the validated enclosure of the solution for one full revolution of the validated enclosure produced by AWA and COSY-VI. It is seen that shortly before completion of the revolution, the enclosures produced by AWA grow rapidly.

We now study the Taylor model representation of the solution obtained by COSY-VI after one revolution; the x_1 component has the form

```
  I  COEFFICIENT              ORDER EXPONENTS
  1  0.9999999999999984         0    0 0   0
  2  0.4999999999999905E-01     1    1 0   0
  3  0.1593548596541794         1    0 1   0
  4  0.2987903618516347E-02     2    2 0   0
  5  0.7967742982712876E-02     2    1 1   0
  6  0.1745863785260356E-01     2    0 2   0
  7  0.4979839364191599E-04     3    3 0   0
  8  0.5551021321878651E-03     3    2 1   0
  9  0.6348634117324201E-03     3    1 2   0
 10  0.1191291278992926E-02     3    0 3   0
 11  0.3258832737620100E-05     4    4 0   0
 12  0.3241341695678232E-06     4    3 1   0
 13  0.3862783715688610E-04     4    2 2   0
 14  0.2689662978922477E-05     4    1 3   0
 15  0.3564904362283420E-04     4    0 4   0
        .....
171  0.1136167325983013E-18    18    117  0
```

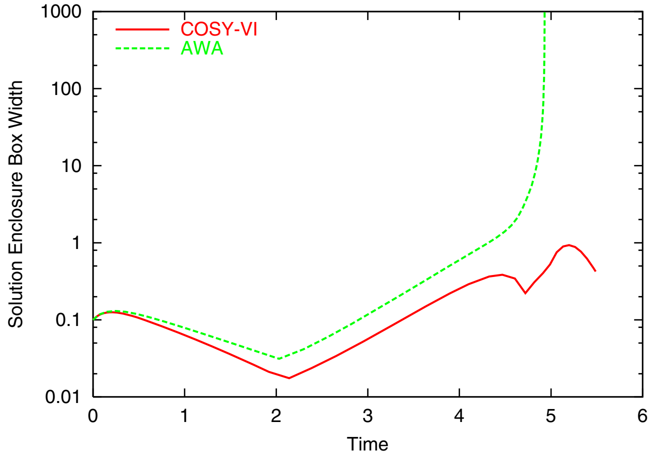


Fig. 2. The width of validated enclosure for solution of the Volterra equation determined by AWA and COSY-VI

```
R   [-.4707095747144810E-010,0.4699004714805186E-010]
```

It can be seen that the zeroth order term is reproduced nearly exactly, as necessary after one revolution. Also the dependence on the first variable is nearly as in the original Taylor model. However, there is now an additional dependence on the second variable, which is significantly larger than the dependence on the first variable, and which induces a significant shearing of the solution. There are also higher order dependencies on initial variables; up to machine precision, terms of up to order 18 contribute, and some of the second order contributions indeed have a magnitude similar to the first order contribution, an indication of strongly nonlinear behavior.

The dependence on the second variable and the higher order contributions are the reason why the box enclosure produced by COSY-VI shown in figure 2 is larger at the end of the integration than it was in the beginning. To determine how much of this is actual overestimation, we insert the Taylor models representing the flow at time t into the invariant in eq. (2.2) of the ODE and subtract from it the value of the invariant at time 0. To the extent the Taylor models represent the true solution, the coefficients of the resulting polynomial should vanish. The bound of the resulting Taylor model is a measure for the sharpness of the approximation.

As a specific example, we show the resulting Taylor model of the invariant defect after one full revolution.

```
I  COEFFICIENT            ORDER EXPONENTS
1  0.1214306433183765E-16   0    0 0  0
2  -.1100465205072787E-16   1    1 0  0
3  -.4109126233720062E-16   1    0 1  0
4  0.3169597288625592E-17   2    2 0  0
5  0.2035589578841535E-16   2    1 1  0
```

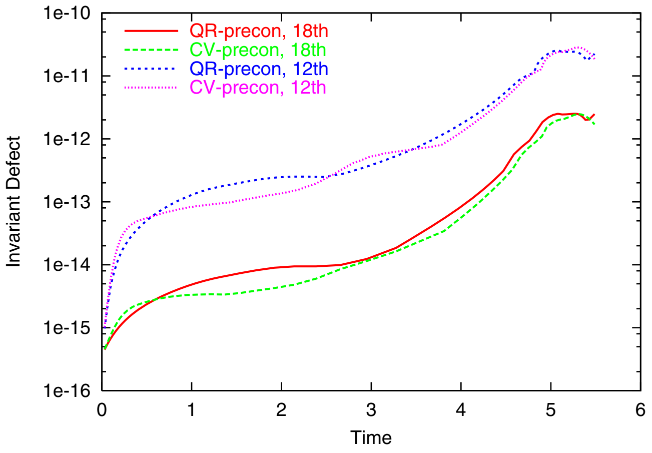


Fig. 3. Invariant defect of the Taylor model integration of the Volterra equation for one revolution for oders 12 and 18 and curvilinear (CV) and QR preconditioning

```
  6  0.2318159770045569E-16   2   0 2  0
  7  -.3702063375093326E-18   3   3 0  0
  8  -.2109853192492055E-17   3   2 1  0
  9  0.7358175212798107E-17   3   1 2  0
 10  0.2956745849914467E-16   3   0 3  0
          .....
 76  0.2469510337886316E-19  15   6 9  0
 R   [-.1689839352652954E-011,0.1691154903051651E-011]
```

Indeed all remaining coefficients are very small, and the remaining terms are just of the magnitude of machine precision. The main contribution is in fact the remainder term of the Taylor model evaluation of the invariant of magnitude around 10^{-12}, which is similar to that of the Taylor model solution after one revolution. Overall, this study shows that the original domain box of width 10^{-1} could be transported for one revolution with an actual overestimation of only around 10^{-12}.

Figure 3 shows in detail the size of the invariant defect as a function of integration time for up to one revolution. Shown are computation orders 18 and 12 and curvilinear (CV) as well as QR preconditioning. It can be seen that the method of order 18 produces an overestimation of around 10^{-12} after one revolution; after a fast ramp-up away from the floating point error floor, a plateau is reached, until the error again increases because the system of ODEs enters a region of strong nonlinearity. On the other hand, figure 2 shows that AWA already early on exhibits relative overestimation of about 2 and then fails before $t = 5$.

In order to assess the long-term behavior of the integration, it is important to first consider some of the specific properties of the Volterra equations. As can be seen from the one-revolution Taylor model enclosure of the solution, one of the important features of the ODEs is that the revolution period strongly depends on the original position on the invariant curves of the Volterra equations. This entails that when integrating the flow of an initial condition box of significant size, some of its corners will lag more and more behind some of the others, and the box will become more and more elongated. Analyzing the one-revolution Taylor model, one sees that within only about five revolutions, the image of the original box has a size similar to the entire parameter space reached during the revolution; thus simplistic long-term integration of this ODE is not possible without further modifications.

One way to treat this problem is to assess the dynamics in terms of a Poincare section, a method frequently employed in the study of long-term behavior (see for example [5]). Here however, we will restrict our attention to a more immediate tool for assessing the long-term behavior, namely repeated forward-backward integration. This approach maintains the nonlinear effects of the problem while away from initial conditions, but avoids the "lag" problem because after one forward-backward cycle, all initial conditions return to their original values.

In the following we assess the long-term forward-backward integration of the Volterra equation using the shrink wrap approach utilized in COSY-VI [13] [9]. Roughly speaking, in this approach the interval remainder bounds are \absorbed" back into the range of the Taylor polynomial of the flow by slightly enlarging the corresponding coefficients. Thus the remaining interval dependencies and the associated risk of eventual blowup disappear. If the intervals to be absorbed into the range are sufficiently small in each step, the increase in the size of the coefficients will also be small. The quality of the invariant over suitable integration times suggests that this is indeed the case.

In figure 4 we show the results of some longer term integration of 100 forward-backwards cycles. The pictures show the width of the solution enclosure box as a function of cycles. The left picture shows the situation for five full cycles; while the box width varies greatly, namely by nearly two orders of magnitude, over one forward-backward cycle it returns to nearly the previous status. The repeated pattern of five very similar looking box widths is visible; furthermore, within each of the five cycles, the widths are mirror symmetric around the middle, which corresponds with the turn-around point.

The right picture in figure 4 shows the situation from cycle 95 to cycle 100. The remarkable fact is that while the curves have significant fine structure, there is no difference discernible to the naked eye; hence the transport after nearly 100 cycles looks almost the same as in the beginning, although the box itself was relatively large and got enhanced to a width of nearly one in each of the forward and backward passes.

3 Autonomous Linear Problems

While of rather limited practical significance, linear autonomous problems are of theoretical interest in the study of integration schemes because much is known about the asymptotic behavior of error growth of common methods. Thus it is interesting to study the behavior of validated integration schemes for such cases. Of particular interest are cases that have eigenvalues of varying magnitudes, since for several validated ODE integration schemes asymptotically this leads to difficulties with ill-conditioned linear algebra operations.

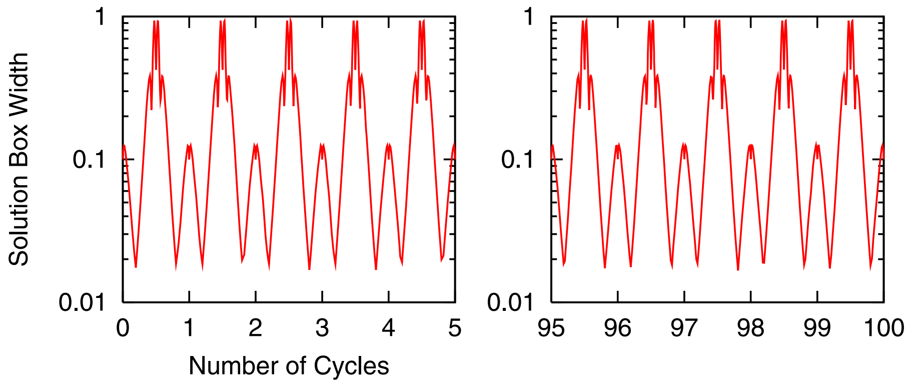


Fig. 4. Evolution of box widths during forward-backward cycles of the Volterra equation. Shown are the situation for the first five cycles as well as cycles 95 to 100

To asses the behavior of the methods, we study three test cases for linear ODEs $x' = Bx$ originally proposed by Markus Neher. We compare the performance of COSY-VI with that of the code AWA. This comparison is interesting because AWA can easily handle these cases since different from the situation in the previous section, the dependence on initial conditions is always linear, and thus COSY-VI's advantage to handle higher order dependence on initial condition is irrelevant. But they are challenging for COSY-VI because only first order terms in initial conditions appear, resulting in extreme sparsity in the Taylor model data structure on computers.

We show results of various computation modes with COSY-VI, namely QR preconditioning (PC-QR), curvilinear preconditioning (PC-CV), blunted parallelepiped preconditioning (PC-BL), and blunted shrink wrapping (SW-BL). Both codes use automatic step size control. COSY-VI uses order 17. AWA uses order 20, and the modes 1 through 4; frequently the mode 4, the \intersection of QR decomposition and interval-vector" performs best. All runs were performed in the arithmetic environment of a 450MHz Pentium III processor. Integration was performed until $t = 1000$ or until failure for the initial box

$$(1,1,1) + 10^{-6} \cdot [-1,1]^3.$$

As a first example, we study a pure contraction with three distinct eigenvalues and obtained the following result of performance.

$$B_1 = \begin{pmatrix} -0.6875 & -0.1875 & 0.08838834762 \\ -0.1875 & -0.6875 & 0.08838834762 \\ 0.08838834762 & 0.08838834762 & -0.875 \end{pmatrix} \approx \begin{pmatrix} -\frac{1}{2} & 0 & 0 \\ 0 & -\frac{3}{4} & 0 \\ 0 & 0 & -1 \end{pmatrix}$$

Mode	t max	Steps	Width
AWA	1000	1216	1.4e-35
VI PC-QR	1000	1633	3.1e-38
VI PC-CV	1000	1463	4.1e-38
VI PC-BL	1000	1620	3.8e-36

VI SW-BL	1000	1726	6.3e-36

We note that the sweeping variable in COSY's Taylor model arithmetic [2], which controls the size of terms retained as significant, has been set to 10^{-40} for this case. Running with a larger value for the sweeping variable leads to a solution set with a size of roughly that larger size.

As the next example, we study a pure rotation

$$B_2 = \begin{pmatrix} 0 & -0.7071067810 & -0.5 \\ 0.7071067810 & 0 & 0.5 \\ 0.5 & -0.5 & 0 \end{pmatrix} \approx \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Mode	t max	Steps	Width
AWA	1000	2549	3.5e-6
VI PC-QR	1000	2021	3.5e-6
VI PC-CV	1000	2046	3.5e-6
VI PC-BL	1000	2021	3.5e-6
VI SW-BL	1000	2030	3.5e-6

Finally we study a combination of a contraction with a rotation

$$B_3 = \begin{pmatrix} -0.125 & -0.8321067810 & -0.3232233048 \\ 0.5821067810 & -0.125 & 0.6767766952 \\ 0.6767766952 & -0.3232233048 & -0.25 \end{pmatrix} \approx \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} \end{pmatrix}$$

Mode	t max	Steps	Width
AWA	1000	3501	3.0e-6
VI PC-QR	1000	2772	3.0e-6
VI PC-CV	1000	2769	3.0e-6
VI PC-BL	1000	2746	4.7e-6
VI SW-BL	1000	2728	1.2e-5

4 Conclusion

Summarizing the results of the numerical experiments for various nonlinear and linear problems shows the following results:

- As expected, the ability to treat higher order dependences on initial conditions leads to a significant performance advantage for COSY-VI for nonlinear problems and larger initial condition domain boxes
- For extended computation times in the Volterra equations, curvilinear preconditioning of Taylor model integration behaves similar to QR preconditioning, and both of them behave significantly better than the AWA approach

- Shrink wrapping allows extended integration periods; over 100 forward-backward cycles through the Volterra equation, growth of box width is not discernible within printer resolution even for rather large boxes where AWA cannot complete a single forward integration
- For linear autonomous problems, the COSY-VI preconditioning methods based on QR, curvilinear, and blunted parallelepiped, all show qualitatively the same behavior as the QR mode of AWA. The latter is known to have error growth similar to the non-validated integration for autonomous linear ODEs. Thus we observe that the three modes of COSY-VI achieve the same type of error growth.
- The number of integration steps of all methods are rather similar.

References

1. K. Makino and M. Berz. Efficient control of the dependency problem based on Taylor model methods. *Reliable Computing*, 5(1):3–12, 1999.
2. K. Makino and M. Berz. Taylor models and other validated functional inclusion methods. *International Journal of Pure and Applied Mathematics*, 6,3:239–316, 2003. available at http://bt.pa.msu.edu/pub.
3. M. Berz and K. Makino. Verified integration of ODEs and flows using differential algebraic methods on high-order Taylor models. *Reliable Computing*, 4(4):361–369, 1998.
4. K. Makino. *Rigorous Analysis of Nonlinear Motion in Particle Accelerators*. PhD thesis, Michigan State University, East Lansing, Michigan, USA, 1998. Also MSUCL-1093.
5. M. Berz. *Modern Map Methods in Particle Beam Physics*. Academic Press, San Diego, 1999. Also available at http://bt.pa.msu.edu/pub.
6. K. Makino and M. Berz. Perturbative equations of motion and differential operators in non-planar curvilinear coordinates. *International Journal of Applied Mathematics*, 3,4:421–440, 2000.
7. M. Berz and K. Makino. Preservation of canonical structure in nonplanar curvilinear coordinates. *International Journal of Applied Mathematics*, 3,4:401–419, 2000.
8. M. Berz. Arbitrary order description of arbitrary particle optical systems. *Nuclear Instruments and Methods*, A298:426, 1990.
9. K. Makino and M. Berz. Suppression of the wrapping effect by Taylor model based validated integrators. submitted. Also MSUHEP-40910, available at http://bt.pa.msu.edu/pub.
10. R. J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. In E. Kaucher, U. Kulisch, and C. Ullrich, editors, *Computer Arithmetic: Scientific Computation and Programming Languages*, pages 255–286. Teubner, Stuttgart, 1987.
11. R. J. Lohner. *Einschliessung der Losung gewohnlicher Anfangs- und Randwertaufgaben und Anwendungen*. Dissertation, Fakultat fur Mathematik, Universitat Karlsruhe, 1988.
12. R. J. Lohner. AWA - Software for the computation of guaranteed bounds for solutions of ordinary initial value problems.
13. K. Makino and M. Berz. The method of shrink wrapping for the validated solution of ODEs. Technical Report MSUHEP-20510, Department of Physics and Astronomy, Michigan State University, East Lansing, MI 48824, 2002.
14. W. F. Ames and E. Adams. Monotonically convergent numerical two-sided bounds for a differential birth and death process. In K. Nickel, editor, *Interval Mathematics*, volume 29 of *Lecture Notes in Computer Science*, pages 135–140, Berlin; New York, 1975. Springer-Verlag.
15. R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, 1979.

On the Use of Intervals in Scientific Computing: What Is the Best Transition from Linear to Quadratic Approximation?

Martine Ceberio[1], Vladik Kreinovich[1], and Lev Ginzburg[2,3]

[1] Department of Computer Science, University of Texas
El Paso, TX 79968, USA
{mceberio,vladik}@cs.utep.edu
[2] Department of Ecology and Evolution
State University of New York Stony Brook
NY 11794, USA
lev@ramas.com
[3] Applied Biomathematics, 100 N. Country Road
Setauket, NY 11733, USA

Abstract. In many problems from science and engineering, the measurements are reasonably accurate, so we can use linearization (= sensitivity analysis) to describe the effect of measurement errors on the result of data processing.

In many practical cases, the measurement accuracy is not so good, so, to get a good estimate of the resulting error, we need to take quadratic terms into consideration – i.e., in effect, approximate the original algorithm by a quadratic function. The problem of estimating the range of a quadratic function is NP-hard, so, in the general case, we can only hope for a good heuristic.

Traditional heuristic is similar to straightforward interval computations: we replace each operation with numbers with the corresponding operation of interval arithmetic (or of the arithmetic that takes partial probabilistic information into consideration). Alternatively, we can first diagonalize the quadratic matrix – and then apply the same approach to the result of diagonalization.

Which heuristic is better? We show that sometimes, the traditional heuristic is better; sometimes, the new approach is better; asymptotically, which heuristic is better depends on how fast, when sorted in decreasing order, the eigenvalues decrease.

1 Formulation of the Problem

Need for Data Processing and Indirect Measurements in Scientific Computing. In many areas of science and engineering, we are interested in the value of a physical quantity y that is difficult (or even impossible) to measure directly. Examples may include the amount of a pollutant in a given lake, the distance to a faraway star, etc.

To measure such quantities, we find auxiliary quantities $x_1, \ldots, x_n$ (easier to measure) that are related to y by a known algorithm $y = f(x_1, \ldots, x_n)$. In some cases, the relation between x_i and y is known exactly. In such cases, to estimate y, we measure x_i, and apply the algorithm f to the results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of measuring x_i. As a result, we get an estimate $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ for y.

In many other practical situations, we only know an approximate relation $y \approx \widetilde{f}(x_1, \ldots, x_n)$, with an upper bound ε_f on the accuracy of this approximation:

$$|\widetilde{f}(x_1, \ldots, x_n) - f(x_1, \ldots, x_n)| \le \varepsilon_f.$$

In such cases, to estimate y, we measure x_i, and apply the algorithm $\widetilde{f}$ to the results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of measuring x_i. As a result, we get an estimate $\widetilde{y} = \widetilde{f}(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ for y.

This indirect measurement (data processing) is one of the main reasons why computers were invented in the first place, and one of the main uses of computers is scientific computing.

Need for Error Estimation for Indirect Measurements in Scientific Computing. Measurements are never 100% accurate. The results $\widetilde{x}_i$ of direct measurements are, in general, different from the actual values x_i. Therefore, the estimate $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ is, in general, different from the actual (unknown) value $y = f(x_1, \ldots, x_n)$. What do we know about the error $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y$ of the indirect measurement?

Estimating Errors of Indirect Measurements: Formulation of the Problem. In many cases, we know the upper bounds Δ_i on the measurement errors $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$ of direct measurements. Once we know such an upper bound, we can guarantee that the actual value x_i lies in the interval $\mathbf{x}_i \stackrel{\text{def}}{=} [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$. In this case, if we know the relation $y = f(x_1, \ldots, x_n)$ exactly, then the only information that we have about y is that y belongs to the range $[\underline{r}, \overline{r}] \stackrel{\text{def}}{=} f(\mathbf{x}_1, \ldots, \mathbf{x}_n)$.

In situations when, instead of knowing the exact relation $y = f(x_1, \ldots, x_n)$, we only know:

- the approximate relation $y \approx \widetilde{f}(x_1, \ldots, x_n)$ between x_i and y and
- we know the upper bound ε_f on the accuracy of approximating f by $\widetilde{f}$,

then we can guarantee that y belongs to the interval $[\underline{r} - \varepsilon_f, \overline{r} + \varepsilon_f]$, where $[\underline{r}, \overline{r}] \stackrel{\text{def}}{=} \widetilde{f}(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ is the range of a known algorithmic function $\widetilde{f}(x_1, \ldots, x_n)$ on the "box" $\mathbf{x}_1 \times \ldots \times \mathbf{x}_n$.

In both cases, to find the range of possible values of y, we must find the range $[\underline{r}, \overline{r}]$ of a known algorithmic function f (or $\widetilde{f}$) on the known box.

Comment. In some engineering situations, instead of knowing the guaranteed upper bounds Δ_i on the measurement errors, we only have *estimates* Δ_i of the upper bounds. In such situations, it is still desirable to compute the corresponding range for y – but we can no longer *absolutely* guarantee that the actual value y belong to the resulting range; we can only guarantee it *under the condition* that the estimates are correct.

Interval Computations: A Way to Estimate Errors of Indirect Measurements. Interval computations enable us to either compute the range a given algorithmic function f (or $\widetilde{f}$) on the given box exactly, or at least to provide an enclosure for this range. For the case when $n = 2$ and the function $f(x_1, x_2)$ is one of the standard arithmetic operations

(+, −, multiplication, etc.), there are known explicit formulas for the range of f. For example,

$$[\underline{x}_1, \overline{x}_1] + [\underline{x}_2, \overline{x}_2] = [\underline{x}_1 + \underline{x}_2, \overline{x}_1 + \overline{x}_2].$$

These formulas form *interval arithmetic*; see, e.g., [3,4,7].

One way to compute the range for more complex functions f is to use straightforward ("naive") interval computations, i.e., replace each operation forming the algorithm f with the corresponding operation from interval arithmetic. This technique leads to an interval that is guaranteed to be an enclosure, i.e., to contain the desired range, but it is known that this interval contains *excess width*, i.e., is wider than the desired range [3,4,7]. How can we reduce this excess width?

When Measurement Errors Are Small, Linearization Works Well. When the measurement errors Δx_i are relatively small, we can expand f into Taylor series in terms of Δx_i and ignore quadratic and higher terms – i.e., keep only linear terms. In a linear expression $f = a_0 + a_1 \cdot \Delta x_1 + \ldots + a_n \cdot \Delta x_n$, each variable $\Delta x_i \in [-\Delta_i, \Delta_i]$ occurs only once. It is known that for such *single-use expressions* (SUE), straightforward interval computations leads to the exact range; see, e.g., [2,3].

Quadratic Approximation Is More Difficult to Analyze. In many real-life situations, measurement errors Δx_i are not so small, so we must also take into consideration terms that are quadratic in Δx_i. So, we must be able to estimate the range of a quadratic function

$$f = a_0 + \sum_{i=1}^{n} a_i \cdot \Delta x_i + \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} \cdot \Delta x_i \cdot \Delta x_j,$$

or, equivalently,

$$f = a_0 + \sum_{i=1}^{n} a_i \cdot \Delta x_i + \sum_{i=1}^{n} a_{ii} \cdot (\Delta x_i)^2 + \sum_{i=1}^{n} \sum_{j \neq i} a_{ij} \cdot \Delta x_i \cdot \Delta x_j. \tag{1.1}$$

There exist methods for computing the exact range of such a function (see, e.g., [4]), but all such methods require 2^n steps – the number of steps which, even for a realistically large number of inputs $n \approx 10^2 - 10^3$, can be impossibly large. Since the problem of estimating range of a given quadratic function is, in general, NP-hard (see, e.g., [6,9]), we cannot hope to get an algorithm that is always faster. So, for large n, we can only compute enclosures.

Two Natural Approaches to Compute Enclosure: Which Is Better? One approach to computing the enclosure of a quadratic approximation function (1.1) is to use naive (straightforward) interval computations. As we have mentioned, in this approach, we often get excess width.

There is a particular case when we do not have any excess width – when the matrix $A = (a_{ij})_{i,j}$ is diagonal. In this case, f can be represented as a sum of the terms $a_i \cdot \Delta x_i + a_{ii} \cdot \Delta x_i^2$ corresponding to different variables, and each of these terms can be reformulated as a SUE expression $a_{ii} \cdot (\Delta x_i + a_i/(2a_{ii}))^2 + \text{const}$ – thus making the whole expression SUE.

Every quadratic function can be represented in a similar diagonal form – as a linear combination of squares of eigenvectors. It therefore seems reasonable to first represent a quadratic function in this form, and only then apply straightforward interval computations.

A natural question is: which approach is better? If none of them is *always* better, then when is the first approach better and when is the second one better?

Beyond Interval Computations: Towards Joint Use of Probabilities and Intervals in Scientific Computing. In many cases, in addition to the upper bounds on Δx_i, we have partial information on the probabilities of different values of $\Delta x \stackrel{\text{def}}{=} (\Delta x_1, \ldots, \Delta x_n)$.

In particular, in some applications, we know that the input variables x_i are not truly independent and are in fact correlated. This knowledge about correlation is also usually represented in the probabilistic terms, as partial information about the probability distribution of Δx.

In all such cases, in addition to the interval range, we would like to compute the information about the probabilities of different values of y. There exist ways of extending interval arithmetic to such cases; see, e.g., [1]. We can therefore use both approaches in these cases as well.

What We Are Planning to Do. In this paper, we show that which method is better depends on the eigenvalues of the matrix $B = (a_{ij} \cdot \Delta_i \cdot \Delta_j)_{i,j}$: on average, the eigenvector method is better if and only if the eigenvalues (when sorted in decreasing order) decrease fast enough.

2 Formalizing the Problem in Precise Terms

Simplifying the Problem. Let us start by simplifying the above problem.

In the original formulation of the problem, we have parameters a_0, a_i, and a_{ij} that describe the function f and the parameters Δ_i that describe the accuracy of measuring each of n variables. We can reduce the number of parameters if we re-scale each of n variables in which a way that Δ_i becomes 1. Indeed, instead of the variables Δx_i, let us introduce the new variables $y_i \stackrel{\text{def}}{=} \Delta x_i / \Delta_i$. For each of y_i, the interval of possible values is $[-1, 1]$. Substituting $\Delta x_i = \Delta_i \cdot y_i$ into the expression (1.1), we get the expression for f in terms of y_i:

$$f = b_0 + \sum_{i=1}^{n} b_i \cdot y_i + \sum_{i=1}^{n} b_{ii} \cdot y_i^2 + \sum_{i=1}^{n} \sum_{j \neq i} b_{ij} \cdot y_i \cdot y_j, \tag{2.2}$$

where $b_0 \stackrel{\text{def}}{=} a_0$, $b_i \stackrel{\text{def}}{=} a_i \cdot \Delta_i$, and $b_{ij} \stackrel{\text{def}}{=} a_{ij} \cdot \Delta_i \cdot \Delta_j$.

In the following text, we will therefore assume that $\Delta_i = 1$ and that the quadratic form has the form (2.2).

Explicit Expressions for the Results of the Two Compared Methods. Let us explicitly describe the results of applying the two methods to the quadratic form (2.2).

If we directly apply straightforward interval computations to the original expression (2.2), then, since $y_i \in [-1,1]$, we get the enclosure $f^{(0)} + \mathbf{f}^{(1)} + \mathbf{f}^{(2)}_{\text{orig}}$, where $f^{(0)} = b_0$, $\mathbf{f}^{(1)} = [-\sum\limits_{i=1}^{n} |b_i|, \sum\limits_{i=1}^{n} |b_i|]$, and

$$\mathbf{f}^{(2)}_{\text{orig}} = \sum_{i=1}^{n} (b_{ii} \cdot [0,1]) + \sum_{i=1}^{n} \sum_{j \neq i} |b_{ij}| \cdot [-1,1]. \tag{2.3}$$

Alternatively, we can represent the matrix $B = (b_{ij})_{i,j}$ in terms of its eigenvalues λ_k and the corresponding unit eigenvectors $e_k = (e_{k1}, \ldots, e_{kn})$, as

$$b_{ij} = \sum_{k=1}^{n} \lambda_k \cdot e_{ki} \cdot e_{kj}. \tag{2.4}$$

In this case, the original expression (2.2) takes the form

$$b_0 + \sum_{i=1}^{n} b_i \cdot y_i + \sum_{k=1}^{n} \lambda_k \cdot \left(\sum_{i=1}^{n} e_{ki} \cdot y_i \right)^2. \tag{2.5}$$

Since $y_i \in [-1,1]$, we conclude that $\sum\limits_{i=1}^{n} e_{ki} \cdot y_i \in [-B_k, B_k]$, where $B_k \overset{\text{def}}{=} \sum\limits_{i=1}^{n} |e_{ki}|$. Therefore, $\left(\sum\limits_{i=1}^{n} e_{ki} \cdot y_i \right)^2 \in [0, B_k^2]$, and so, when applied to the expression (2.5), straightforward interval computations lead to the expression $f^{(0)} + \mathbf{f}^{(1)} + \mathbf{f}^{(2)}_{\text{new}}$, in which linear terms $f^{(0)}$ and $\mathbf{f}^{(1)}$ are the same, while

$$\mathbf{f}^{(2)}_{\text{new}} = \sum_{k=1}^{n} \lambda_k \cdot \left[0, \left(\sum_{i=1}^{n} |e_{ki}| \right)^2 \right]. \tag{2.6}$$

So, to decide which method is better, it is sufficient to consider only quadratic terms.

Example When the Eigenvalue-Related Expression Is Better. If the matrix B has only one non-zero eigenvector $\lambda_1 \neq 0$, then the formula (2.5) takes a simplified form: $\lambda_1 \cdot (\sum\limits_{i=1}^{n} e_{1i} \cdot y_i)^2$. This is a SUE expression, so straightforward interval computations lead to the exact range.

For such matrices, the original expression (1) is not necessarily SUE, and may lead to excess width. For example, for a 2×2 matrix with $b_{ij} = 1$ for all i and j, the only non-zero eigenvalue is $\lambda_1 = 2$ (with eigenvector $(1,1)$). So, the new expression leads to the exact range $[0,4]$. On the other hand, if we apply straightforward interval computations to the original expression (2.2), then the resulting expression (2.3) leads to $[-2,4]$, i.e., to excess width.

Example When the Original Expression Is Better. For the identity matrix B, the original quadratic expression (2.2) leads to a SUE expression $\sum b_{ii} \cdot (\Delta x_i)^2$ for which straightforward interval computations lead to the exact range. For example, for $n = 1$, we get the range $[0, 2]$.

On the other hand, if we select eigenvectors that are different from $(1, 0)$ and $(0, 1)$, we may get excess width. For example, if we choose $e_1 = (\sqrt{2}/2, \sqrt{2}/2)$ and $e_2 = (\sqrt{2}/2, -\sqrt{2}/2)$, then, for straightforward interval computations, the range of $\frac{\sqrt{2}}{2}\Delta x_1 + \frac{\sqrt{2}}{2}\Delta x_2$ is $[-\sqrt{2}, \sqrt{2}]$, hence the range of its square is $[0, 2]$, and the range of the resulting quadratic expression is estimated as $[0, 4]$.

How Do We Compare Different Approaches: Randomization Needed. The main difference between the two cases is in the eigenvalues of the matrix B: In the first example, we had only one non-zero eigenvalue, and the eigenvalue-related expression leads to better estimates. In the second example, we have equal eigenvalues, and the original expression is better. It is therefore natural to assume that which method is better depends on the eigenvalues λ_k of the matrix B.

We should not expect a result of the type "if we have certain λ_k, then the first method is always better" – which method is better depends also on the eigenvectors. For example, in the second case, if we select $(1, 0)$ and $(0, 1)$ as eigenvectors, then the eigenvalue-related expression also leads to the same optimal range estimate. In other words, for a given set of eigenvalues λ_k, we should not expect a result saying that one of the methods is better for *all* possible eigenvectors: for some eigenvectors the first methods will be better, for some others the second method will be better. In such a situation, it is reasonable to analyze which method is better *on average*, if we consider random eigenvectors.

Natural Probability Measure on the Set of All Eigenvectors. What is the natural probability measure on the set of all possible eigenvectors $e_1, \ldots, e_n$? In general, we have n mutually orthogonal unit vectors, i.e., an orthonormal base in the n-dimensional space. It is reasonable to assume that the probability distribution on the set of all such bases is rotation-invariant. This assumption uniquely determines the probability distribution; see, e.g., [5,8].

Indeed, the first unit vector e_1 can be uniquely represented by its endpoint on a unit sphere. The only possible rotation-invariant distribution on a unit sphere is a uniform distribution. Once e_1 is fixed, e_2 can be any vector from a sphere in an $(n-1)$-dimensional space of all vectors orthogonal to e_1; the only rotation-invariant distribution on this sphere is also uniform, etc. So, in the resulting distribution, e_1 is selected from the uniform distribution on the unit sphere, e_2 from the uniform distribution on the unit sphere in the subspace of all vectors $\perp e_1$, etc.

3 Main Result

Theorem 1. *When $n \to \infty$, then asymptotically, the expected values are:*

$$E[\mathbf{f}^{(2)}_{\text{orig}}] \sim \left[-\sqrt{\frac{2}{\pi}} \cdot n \cdot \sqrt{\sum_{k=1}^{n} \lambda_k^2}, \sqrt{\frac{2}{\pi}} \cdot n \cdot \sqrt{\sum_{k=1}^{n} \lambda_k^2}\right]; \tag{3.7}$$

$$E[\mathbf{f}^{(2)}_{\text{new}}] \sim \left[\frac{2}{\pi} \cdot n \cdot \sum_{k:\lambda_k<0} \lambda_k, \frac{2}{\pi} \cdot n \cdot \sum_{k:\lambda_k>0} \lambda_k\right]. \tag{3.8}$$

Conclusions. If $\sum |\lambda_k| < \sqrt{\pi/2} \cdot \sqrt{\sum \lambda_k^2}$, then asymptotically, $E[\mathbf{f}^{(2)}_{\text{new}}] \subset E[\mathbf{f}^{(2)}_{\text{orig}}]$, so the eigenvector-based method is definitely better.

If $\sum |\lambda_k| < \sqrt{2\pi} \cdot \sqrt{\sum \lambda_k^2}$, then the interval $E[\mathbf{f}^{(2)}_{\text{new}}]$ is narrower than $E[\mathbf{f}^{(2)}_{\text{orig}}]$, so in this sense, the new method is also better.

Example. The spectrum λ_k often decreases according to the power law $\lambda_k \sim k^{-\alpha}$. In this case, $\sum |\lambda_k| \approx \int_1^\infty x^{-\alpha}\,\mathrm{d}x = 1/(\alpha-1)$ and $\sum \lambda_k^2 \approx \int_1^\infty x^{-2\alpha}\,\mathrm{d}x = 1/(2\alpha-1)$, so the above inequality turns into $(\alpha-1)^2 \geq (2/\pi) \cdot (2\alpha-1)$, which is equivalent to

$$\alpha \geq 1 + \frac{2}{\pi} + \sqrt{\left(1+\frac{2}{\pi}\right) \cdot \frac{2}{\pi}} \approx 2.7. \tag{3.9}$$

Hence, if the eigenvalues decrease fast ($\alpha \geq 2.7$), the new method is definitely better. For $\alpha \geq 1.6$, the new method leads to narrower intervals; otherwise, the traditional method leads, on average, to better estimates.

Proof of the Theorem. Before we start the proof, let us derive some auxiliary formulas. Since each vector e_k is a unit vector, we have $\sum\limits_i e_{ki}^2 = 1$. Due to rotation invariance, the expected value $E[e_{ki}^2]$ should not depend on i, hence $E[e_{ki}^2] = 1/n$. Similarly, from $\sum\limits_i e_{ki} \cdot e_{li} = 0$ and rotation invariance, we conclude that $E[e_{ki} \cdot e_{li}] = 0$.

For given k, l, and $i \neq j$, the value $E[e_{ki} \cdot e_{lj}]$ should not change under the transformation $x_i \to x_i$ and $x_j \to -x_j$, so $E[e_{ki} \cdot e_{lj}] = 0$.

To compute $E[\mathbf{f}^{(2)}_{\text{orig}}]$, we must find $E[b_{ii}]$ and $E[|b_{ij}|]$. By definition (2.4), for each i, $E[b_{ii}] = \sum\limits_k \lambda_k \cdot E[e_{ki}^2] = (1/n) \cdot \sum \lambda_k$, so the sum of n such terms is proportional to $\sum \lambda_k$.

For $i \neq j$, due to the central limit theorem, the distribution for b_{ij} (formula (2.4)) is asymptotically Gaussian, so asymptotically, $E[|b_{ij}|] \sim \sqrt{2/\pi} \cdot \sqrt{E[b_{ij}^2]}$. Here, $E[b_{ij}^2] = \sum\limits_k \sum\limits_l \lambda_k \cdot \lambda_l \cdot E[e_{ki} \cdot e_{kj} \cdot e_{li} \cdot e_{lj}]$. Due to symmetry, each $k \neq l$ term is 0, so $E[b_{ij}^2] = \sum\limits_k \lambda_k^2 \cdot E[e_{ki}^2 \cdot e_{kj}^2]$. Asymptotically, e_{ki} and e_{kj} are independent, so $E[e_{ki}^2 \cdot e_{kj}^2] \sim E[e_{ki}^2] \cdot E[e_{kj}^2] = (1/n)^2$. Therefore, $E[b_{ij}^2] \sim (1/n)^2 \cdot \sum \lambda_k^2$, hence $E[|b_{ij}|] \sim \sqrt{2/n} \cdot (1/n) \cdot \sqrt{\sum \lambda_k^2}$. The sum of $n(n-1)$ such terms is $\sim \sqrt{2/\pi} \cdot n \cdot \sqrt{\lambda_k^2}$. The sum of the terms $E[b_{ii}]$ is asymptotically smaller, so when $n \to \infty$, we get the expression (3.7).

For the new expression, we must compute, for every k, the expected value of

$$E\left[\left(\sum_i |e_{ki}|\right)^2\right] = \sum_{i,j} E[|e_{ki}| \cdot |e_{kj}|].$$

Asymptotically, e_{ki} and e_{kj} are independent, and $E[|e_{ki}|] \sim \sqrt{2/\pi} \cdot \sqrt{E[e_{ki}^2]} = \sqrt{2/\pi} \cdot (1/\sqrt{n})$. Thus, the sum of all the terms $i \neq j$ is $\sim n^2 \cdot (2/\pi) \cdot (1/n) = (2/\pi) \cdot n$. The terms with $i = j$ are asymptotically smaller, so we get the desired expression (3.8). □

Acknowledgments

This work was supported in part by NASA grant NCC5-209, by the Air Force Office of Scientific Research grant F49620-00-1-0365, by NSF grants EAR-0112968, EAR-0225670, and EIA-0321328, and by the Army Research Laboratories grant DATM-05-02-C-0046. This work was partly performed during V. Kreinovich's visit to Brazil. This visit was sponsored by the Brazilian funding agency CTINFO/CNPq.

The authors are thankful to the anonymous referees for valuable suggestions.

References

1. Ferson, S.: RAMAS Risk Calc 4.0, CRC Press, Boca Raton, Florida, 2002.
2. Hansen, E.: Sharpness in interval computations, Reliable Computing **3** (1997) 7–29.
3. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis: With Examples in Parameter and State Estimation, Robust Control and Robotics, Springer, London, 2001.
4. Kearfott, R.B.: Rigorous Global Search: Continuous Problems. Kluwer, Dordrecht, 1996.
5. Koltik, E., Dmitriev, V.G., Zheludeva, N.A., Kreinovich, V.: An optimal method for estimating a random error component, Investigations in Error Estimation, Proceedings of the Mendeleev Metrological Institute, Leningrad, 1986, 36–41 (in Russian).
6. Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: Computational Complexity and Feasibility of Data Processing and Interval Computations, Kluwer, Dordrecht, 1997.
7. Moore, R.E.: Methods and Applications of Interval Analysis. SIAM, Philadelphia, 1979.
8. Trejo, R., Kreinovich, V.: Error Estimations for Indirect Measurements: Randomized vs. Deterministic Algorithms For "Black-Box" Programs, In: Rajasekaran, S., Pardalos, P., Reif, J., Rolim, J., eds., Handbook on Randomized Computing, Kluwer, 2001, 673-729.
9. Vavasis, S.A.: Nonlinear Optimization: Complexity Issues, Oxford University Press, New York, 1991.

HPC-ICTM: The Interval Categorizer Tessellation-Based Model for High Performance Computing

Marilton S. de Aguiar[1], Graçaliz P. Dimuro[1], Fábia A. Costa[1], Rafael K.S. Silva[2], César A.F. De Rose[2], Antônio C.R. Costa[1,3], and Vladik Kreinovich[4]

[1] Escola de Informática, Universidade Católica de Pelotas
Rua Felix da Cunha 412, 96010-000 Pelotas, Brazil
{marilton,liz,fabia,rocha}@ucpel.tche.br
[2] PPGCC, Pontifícia Universidade Católica do Rio Grande do Sul
Av. Ipiranga 6681, 90619-900 Porto Alegre, Brazil
{rksilva,derose}@inf.pucrs.br
[3] PPGC, Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves 9500, 91501-970 Porto Alegre, Brazil
[4] Department of Computer Science, University of Texas at El Paso
El Paso, TX 79968, USA
vladik@cs.utep.edu

Abstract. This paper presents the Interval Categorizer Tessellation-based Model (ICTM) for the simultaneous categorization of geographic regions considering several characteristics (e.g., relief, vegetation, land use etc.). Interval techniques are used for the modelling of uncertain data and the control of discretization errors. HPC-ICTM is an implementation of the model for clusters. We analyze the performance of the HPC-ICTM and present results concerning its application to the relief/land-use categorization of the region surrounding the lagoon *Lagoa Pequena* (RS, Brazil), which is extremely important from an ecological point of view.

1 Introduction

The ICTM (*Interval Categorizer Tessellation Model*) is a multi-layered and multi dimensional tessellation model for the simultaneous categorization of geographic regions considering several different characteristics (relief, vegetation, climate, land use etc.) of such regions, which uses interval techniques [4,9] for the modelling of uncertain data and the control of discretization errors.

To perform a *simultaneous categorization*, the ICTM proceeds (in parallel) to individual categorizations considering one characteristic per layer, thus generating different subdivisions of the analyzed region. An appropriate projection procedure of the categorizations performed in each layer into a basis layer provides the final categorization that allows the combined analysis of all characteristics that are taken into consideration by the specialists in the considered application, allowing interesting analyzes about their mutual dependency.

An implementation of the ICTM for the relief categorization of geographic regions, called TOPO-ICTM (*Interval Categorizer Tessellation Model for Reliable Topographic*

Segmentation), performs a bi-dimensional analysis of the declivity of the relief function in just one layer of the model [1]. The data input are extracted from satellite images, where the heights are given in certain points referenced by their latitude and longitude coordinates. The geographic region is represented by a regular tessellation that is determined by subdividing the total area into sufficiently small rectangular subareas, each one represented by one cell of the tessellation. This subdivision is done according to a cell size established by the geophysics or ecology analyst and it is directly associated to the refinement degree of the tessellation. Applications in Geophysics and Ecology were found, where an adequate subdivision of geographic areas into segments presenting similar topographic characteristics is often convenient (see, e.g: [2,5]).

The aim of this paper is to present the `ICTM` model and describe a particular implementation of the model for clusters, called `HPC-ICTM`. We discuss the performance of the `HPC-ICTM` and present some results concerning the application of a 2-layered bi-dimensional model to the *relief/land use* categorization of the region surrounding the lagoon *Lagoa Pequena* (Rio Grande do Sul, Brazil), which is extremely important from an ecological point of view.

The paper is organized as follows. Section 2 presents the `ICTM` model and the categorization process. Some results on categorizations are shown in Sect. 3. The performance of the `HPC-ICTM` is discussed in Sect. 4. Section 5 is the Conclusion.

2 The `ICTM` Model

This section introduces the multi-layered interval categorizer tessellation-based model `ICTM`, formalized in terms of matrix operations, extending the results presented in a previous paper [1][5], for the single-layered model `TOPO-ICTM`.

A *tessellation*[6] is a matrix M, whose each entry[7] at the x-th row and the y-th column is denoted by m_{xy}. For $L \in \mathbb{N}$ and tessellations $M^1, \ldots, M^L$, an *L-layered tessellation* is a structure $\mathcal{M}^L = \left(M^1, \ldots, M^L\right)$, where each entry at the l-th layer, x-th row and y-th column is denoted by m^l_{xy}.

2.1 Using Interval Matrices

In many applications considered by the Geophysics and Ecologists, usually there are too much data to be analyzed, most of which is irrelevant. We take, for each subdivision of the geographic region, the average of the values attached to the points of each layer, which are the entries m^l_{xy} of the L matrices of the tessellation $\mathcal{M}^L$. To simplify the data, we normalize them by dividing each m^l_{xy} by the largest m^l_{max} of these values, obtaining a relative value $rm^l_{xy} = \frac{m^l_{xy}}{|m^l_{max}|}$. The *relative matrix* of a layer l is given by $RM^l = \frac{M^l}{|m^l_{max}|}$.

[5] The proofs are omitted since they are similar to those presented in [1].

[6] To simplify the notation, we consider bi-dimensional tessellations only.

[7] For the application considered in this paper, the entries of the tesselation matrices are all non-negative. However, negative values may also be considered (e.g., when the data coming from the relief are determined with respect to the sea level).

Sometimes we have problems in representing the uncertain data provided by the sources of the considered application. Even if the values associated to the points are pretty accurate (like, for instance, the heights provided by satellite images), we have to deal with the errors that came from the discretization of the area in terms of the discrete set of the tessellation cells. We apply techniques from Interval Mathematics [4,9] to control the errors associated to the cell values. See examples of using intervals in solving similar problems in [2,4,9].

Observe that, considering a layer l, for each $\xi\upsilon$ that is different from xy, it is reasonable to estimate a value $h^l_{\xi\upsilon}$, attached to the point $\xi\upsilon$ in the layer l, as the value rm^l_{xy} at the point xy which is closest to $\xi\upsilon$, meaning that $\xi\upsilon$ belongs to the same segment of area as xy. For each cell xy in the layer l, let Δ^l_x be the largest possible error of the corresponding approximation considering the west-east direction. Then the approximation error ϵ^l_x is bounded by $\epsilon^l_x \leq \Delta^l_x = 0.5 \cdot \min(|rm^l_{xy} - rm^l_{(x-1)y}|, |rm^l_{(x+1)y} - rm^l_{xy}|)$. Now, for each cell xy in the layer l, let Δ^l_y be the largest possible error of the corresponding approximation considering the north-south direction. Therefore, the *approximation error* ϵ^l_y is bounded by $\epsilon^l_y \leq \Delta^l_y = 0.5 \cdot \min(|rm^l_{xy} - rm^l_{x(y-1)}|, |rm^l_{x(y+1)} - rm^l_{xy}|)$.

Thus, considering a given y in the layer l, the intervals $im^{x,l}_{xy} = [m^l_{x^-y}, m^l_{x^+y}]$, where $m^l_{x^-y} = rm^l_{xy} - \Delta^l_x$ and $m^l_{x^+y} = rm^l_{xy} + \Delta^l_x$, contain all the possible values of $h^l_{\xi y}$, for $x - \frac{1}{2} \leq \xi \leq x + \frac{1}{2}$. Similarly, for a fixed x in the layer l, for each y such that $y - \frac{1}{2} \leq \upsilon \leq y + \frac{1}{2}$, it follows that $h^l_{x\upsilon} \in im^{y,l}_{xy} = [m^l_{xy^-}, m^l_{xy^+}, m^l_{xy^-} = rm^l_{xy} - \Delta^l_y$, with $m^l_{xy^+} = rm^l_{xy} + \Delta^l_y$. For each layer l, the *interval matrices* associated to the relative matrix RM^l are denoted by $IM^{\times,l}$ and $IM^{y,l}$.

2.2 The Categorization Process

To obtain a declivity categorization[8] in each layer, we assume that the approximation functions introduced by the model are piecewise linear functions[9]. The model determines a piecewise linear approximation function (and corresponding set of limit points between the resulting sub-regions) that fits the constraints imposed by the interval matrices. To narrow the solution space to a minimum, we take a qualitative approach to the approximation functions, clustering them in equivalence classes according to the signal of their declivity (positive, negative, null). For each layer l, the model obtains the class of approximation functions compatible with the constraints of the interval matrices $IM^{x,l}$ and $IM^{y,l}$.

Proposition 1. *For a given xy in a layer l, it holds that:*

(i) Considering the direction west-east, if $m^l_{x^+y} \geq m^l_{(x+1)^-y}$, then there exists a non-increasing relief approximation function between xy and $(x+1)y$.

[8] This declivity categorization was inspired by [2]. By *declivity* we mean the tangent of the angle α between the approximation function and the positive direction of the horizontal axis. The declivity happens to be continuous, since the approximation functions of the model are total and have no steps. The declivity is positive, negative or null if $0 < \alpha < \frac{\pi}{2}$, $\frac{\pi}{2} < \alpha < \pi$ or $\alpha = 0$, respectively.

[9] Piecewise linear functions were considered for simplicity. A more general approach may consider other kinds of piecewise monotonic functions.

(ii) *Considering the direction west-east, if* $m^l_{(x-1)^- y} \leq m^l_{x^+ y}$, *then there exists a non-decreasing relief approximation function between* $(x-1)y$ *and* xy.

(iii) *Considering the direction north-south, if* $m^l_{xy^+} \geq m^l_{x(y+1)^-}$, *then there exists a non-increasing relief approximation function between* xy *and* $x(y+1)$.

(iv) *Considering the direction north-south, if* $m^l_{x(y-1)^-} \leq m^l_{xy^+}$, *then there exists a non-decreasing relief approximation function between* $x(y-1)$ *and* xy.

Definition 1. *A* declivity register *of an* xy*-cell in a layer* l *is a tuple*

$$dm^l_{xy} = (e^l_{xy}, w^l_{xy}, s^l_{xy}, n^l_{xy}),$$

where e^l_{xy}, w^l_{xy}, s^l_{xy} *and* n^l_{xy}, *called* directed declivity registers *for east, west, south and north directions, respectively, are evaluated according to the conditions considered in Prop. 1:*

(a) *For non border cells:* $e^l_{xy} = 0$, *if (i) holds;* $w^l_{xy} = 0$, *if (ii) holds;* $s^l_{xy} = 0$, *if (iii) holds;* $n^l_{xy} = 0$, *if (iv) holds;* $e^l_{xy} = w^l_{xy} = s^l_{xy} = n^l_{xy} = 1$, *otherwise.*

(b) *For east, west, south and north border cells:* $e^l_{xy} = w^l_{xy} = s^l_{xy} = n^l_{xy} = 0$, *respectively. The other directed declivity registers of border cells are also determined according to item (a).*

The declivity register matrix of the layer l *is the matrix* $dM^l = \left[dm^l_{xy}\right]$.

Corollary 1. *Considering a layer* l *and the west-east direction, any relief approximation function is either (i)* strictly increasing *between* xy *and* $(x+1)y$ *if* $e^l_{xy} = 1$ *(and, in this case,* $w^l_{(x+1)y} = 0$*); or (ii)* strictly decreasing *between* xy *and* $(x+1)y$ *if* $w^l_{(x+1)y} = 1$ *(and, in this case,* $e^l_{xy} = 0$*); or (iii)* constant *between* xy *and* $(x+1)y$ *if* $e^l_{xy} = 0$ *and* $w^l_{(x+1)y} = 0$. *Similar results hold for the north-south direction.*

Associating convenient weights to the directed declivity registers of a cell xy in a layer l, it is possible to obtain a binary encoding that represents the *state* sm^l_{xy} of such cell, given by

$$sm^l_{xy} = 1 \times e^l_{xy} + 2 \times s^l_{xy} + 4 \times w^l_{xy} + 8 \times n^l_{xy}.$$

The *state matrix* of layer l is given by $SM^l = \left[sm^l_{xy}\right]$. Thus, any cell can assume one and only one state represented by the value $sm^l_{xy} = 0..15$.

A *limiting cell* is the one where the relief function changes its declivity, presenting critical points (maximum, minimum or inflection points). According to this criteria, any non-limiting cell should satisfy one of the conditions listed in Table 1. The border cells are assumed to be limiting. To identify the limiting cells, we use a *limiting register* λm^l_{xy} associated to each xy-cell of a layer l, defined as:

$$\lambda m^l_{xy} = \begin{cases} 0 \text{ if one of the conditions listed in Table 1 holds;} \\ 1 \text{ otherwise.} \end{cases} \tag{2.1}$$

The *limiting matrix* of the layer l is $\lambda M^l = \left[\lambda m^l_{xy}\right]$. Analyzing this matrix, we proceed to the subdivision of the whole area into constant declivity categories.

Table 1. Conditions of non limiting cells

Conditions	Conditions
$e^l_{(x-1)y} = e^l_{xy} = 1$	$n^l_{xy} = n^l_{x(y+1)} = 1$
$w^l_{xy} = w^l_{(x+1)y} = 1$	$s^l_{x(y-1)} = s^l_{xy} = 1$
$e^l_{(x-1)y} = e^l_{xy} = w^l_{xy} = w^l_{(x+1)y} = 0$	$s^l_{x(y-1)} = s^l_{xy} = n^l_{xy} = n^l_{x(y+1)} = 0$

Definition 2. *The constant declivity sub-region $\mathcal{SR}^l_{xy}$, associated to a non limiting cell xy in a layer l, is inductively defined: (i) $xy \in \mathcal{SR}^l_{xy}$; (ii) If $x'y' \in \mathcal{SR}^l_{xy}$, then all its neighbor cells that are not limiting cells also belong to $\mathcal{SR}^l_{xy}$.*

Definition 2 leads to a recursive algorithm similar to the ones commonly used to fulfill polygons.

2.3 The Projection Procedure

The basis layer λM^π is used to receive the *projection* of the limiting cells of all layers. This projection is useful for the identification of interesting information, such as: (i) the cells which are limiting in all layers; (ii) the projection of all sub-areas; (iii) the certainty degree of a cell to be limiting etc.

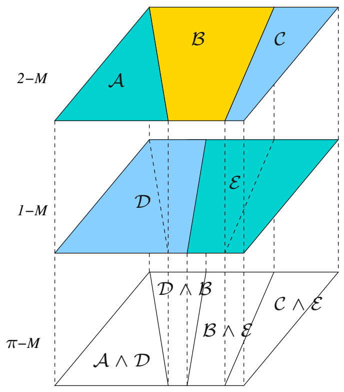
Two projection algorithms were proposed. In the first algorithm ($Type\ I$, Fig. 1), if a certain cell is a limiting cell just in one layer then it will be projected on the basis layer also as a limiting cell. A weight $0 \leq w_l \leq 1$, for $l = 1, \ldots, L$ is associated to each layer, so that $w_l = 1$ ($w_l = 0$) indicates that the layer l is (is not) selected for the projection. Then, the projection of limiting cells on the basis layer is given by $\lambda M^\pi = \bigvee_{l=1}^{L} \lambda M^l \times w_l$. In the second algorithm ($Type\ II$, Fig. 2), each layer may present different degrees of participation in the determination of the projection[10]. In this case, the projection of limiting cells on the basis layer is given by $\lambda M^\pi = \bigvee_{l=1}^{L} \lambda M^l \times \overline{w}_l$, where $\overline{w}_l = \frac{w_l}{\sum_{i=1}^{L} w_i}$ are the normalized weights.

3 Some Results on Categorizations

This section presents the *relief* and *land use* categorizations obtained for the region surrounded the lagoon *Lagoa Pequena* (Rio Grande do Sul, Brazil). These analyzes are to be used for the environment characterization of that region, aiming to give subsidies for its integrated preservation/management.

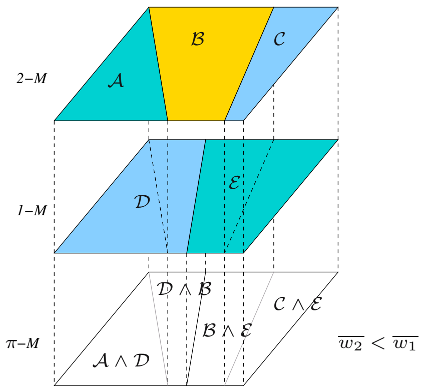
Figure 3 shows the location of the lagoon and a *land use* categorization of the region surrounded it, which shall be combined with relief categorizations. For the portion of

[10] Moreover, the sum of the weights may not be equal to 1 in the case that the analyst does not have a clear perception of the importance of each layer in the whole process.



2–M
1–M
π–M
$\mathcal{A}$
$\mathcal{B}$
$\mathcal{C}$
$\mathcal{D}$
$\mathcal{E}$
$\mathcal{D} \wedge \mathcal{B}$
$\mathcal{C} \wedge \mathcal{E}$
$\mathcal{B} \wedge \mathcal{E}$
$\mathcal{A} \wedge \mathcal{D}$

Fig. 1. $Type\ I$ projection procedure



2–M
1–M
π–M
$\mathcal{A}$
$\mathcal{B}$
$\mathcal{C}$
$\mathcal{D}$
$\mathcal{E}$
$\mathcal{D} \wedge \mathcal{B}$
$\mathcal{C} \wedge \mathcal{E}$
$\mathcal{B} \wedge \mathcal{E}$
$\mathcal{A} \wedge \mathcal{D}$
$\overline{w_2} < \overline{w_1}$

Fig. 2. $Type\ II$ projection procedure

the LANDSAT image[11] shown in Fig. 4(a), the ICTM produced the relief categorization presented in Fig. 4(b), for the Digital Elevation Model (DEM), and in Fig. 4(c), for a 3D visualization of this categorization. Figure 4(c) shows the ICTM relief characterization given in terms of the state and limiting matrices, where a pleistocene marine barrier can be distinguished.

4 Performance Analysis

The parallel implementation of the ICTM model for clusters was done using the library MPI (*Message Passing Interface*) [7]. Each process of the parallel program is responsible for a categorization performed in one of the layers of the model, in a modified *master-slave* schema, where the slave processes receive the information sent by the master process and, after executing their jobs, they generate their own outputs. The master process is responsible for loading the input files and parameters (the data and the radius), sending the radius value for the L slave processes to start the categorization process in each layer. The file system is shared, that is the directory with the input files is accessible by all the cluster's nodes.

For the analysis of the performance, we consider three tessellation matrices: M_1 (241×241), M_2 (577×817) and M_3 (1309×1765). The results were processed by the CPAD (Research Center of High Performance Computing of PUCRS/HP, Brasil), with the following environment: (i) heterogenous cluster with 24 machines: ($e800$) 8 machines with two processors Pentium III 1Ghz and 256MB of memory, ($e60$) 16 machines with two processors Pentium III 550Mhz and 256MB of memory; (ii) Ethernet and Myrinet networks; (iii) MPICH version of MPI Library; (iv) 1 front-end machine with two processors Pentium III 1Ghz and 512MB of memory. Table 2 presents the results of sequential implementation processed in the cluster front-end, and also the results of the parallel implementation of ICTM. As expected, the machine $e800$ is faster then the

[11] The coordinates are Datum SAD69 (South American Datum 1969) and UTM (Universal Transverse Mercator), Zone 22 South Hemisphere.

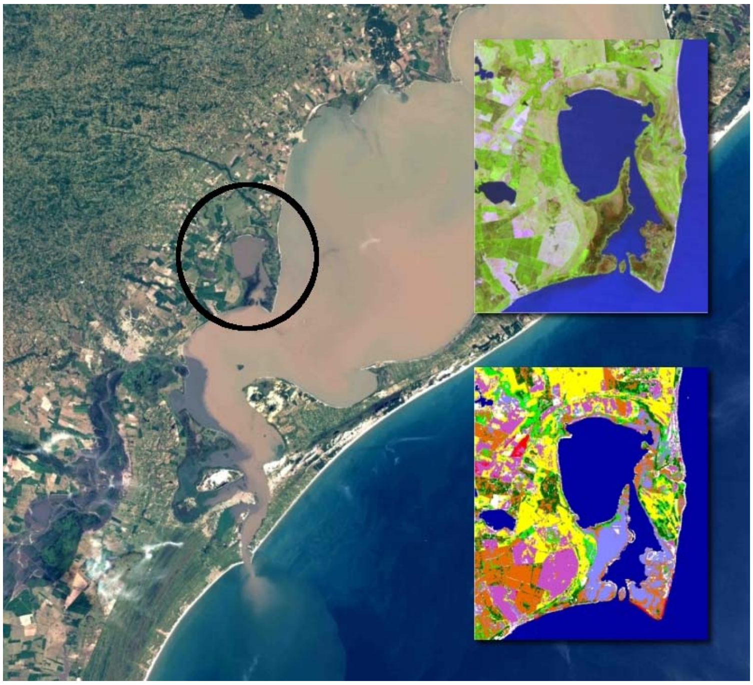


Fig. 3. Land use Map of the region surrounded the *Lagoa Pequena* (light blue: wetland, dark blue: water, yellow: crops and past, purple: transitional, light green: riparia forest, dark green: restinga forest, red: lagoon beaches, white: without classification)

Table 2. Analysis of ICTM results (T_s is the time of sequential implementation and $T_{e60-Eth}$, $T_{e800-Eth}$, $T_{e60-Myr}$, $T_{e800-Myr}$ are the times of parallel implementations)

Matrix	# of Layers	T_s	# of Proc.	$T_{e60-Eth}$	$T_{e800-Eth}$	$T_{e60-Myr}$	$T_{e800-Myr}$
M_1	3	1.947s	4	2.015s	1.340s	1.998s	1.209s
M_1	5	3.298s	6	2.228s	1.510s	2.263s	1.254s
M_2	3	16.041s	4	11.180s	6.038s	12.003s	6.248s
M_2	5	27.015s	6	11.383s	6.194s	12.011s	6.331s
M_3	3	15m50.871s	4	50.277s	27.635s	54.742s	28.982s
M_3	5	35m14.322s	6	50.551s	27.665s	55.657s	29.068s

machine $e60$. However, in some cases, the *ethernet network* was faster than *myrinet network*, due to the low volume of inter-processor communication. In general, the difference between the performance of the two networks is observed when the processors require a lot of message exchanges. Notice that, even when the number of layers increased, a significant time variation was not observed.

One interesting feature of the parallel implementation is the partition of input data, which reduces the amount of memory to be stored in a single processor. We observe

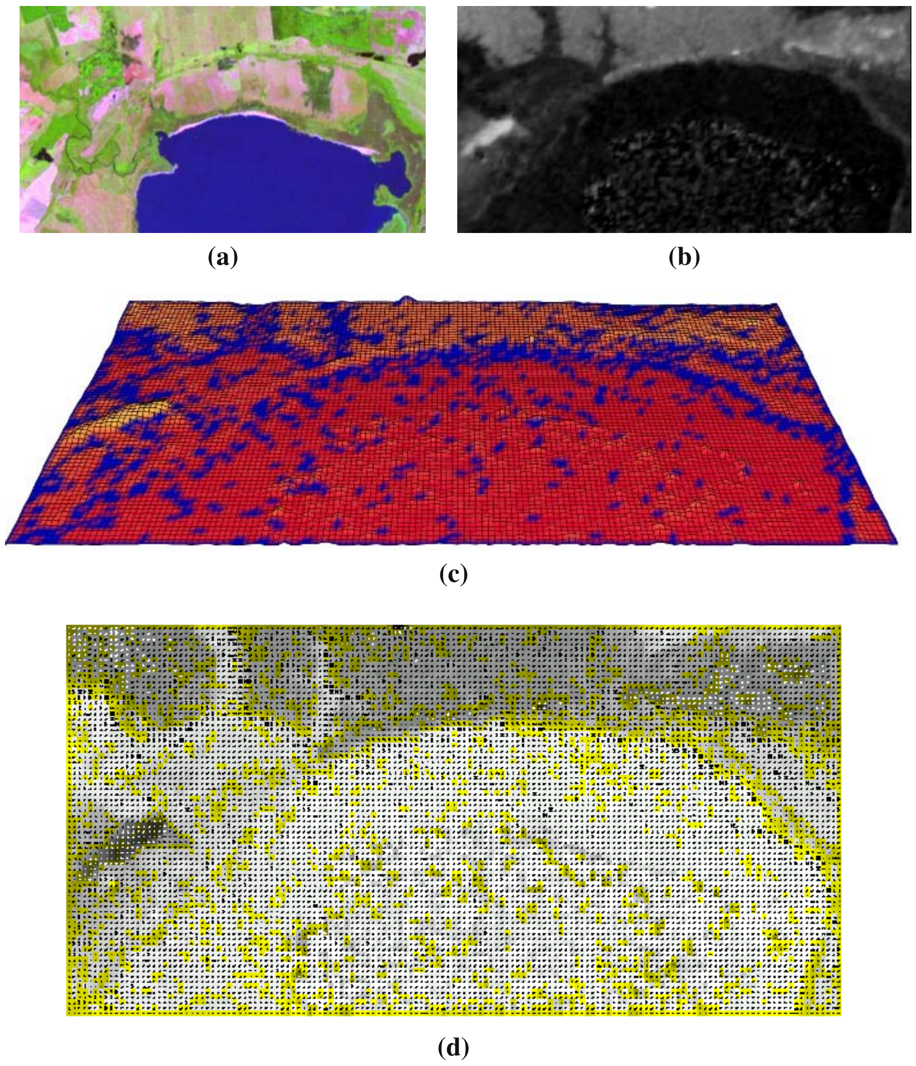


Fig. 4. Relief categorizations of a portion of the region surrounding the *Lagoa Pequena*: (a) LANDSAT image, coordinates: Upper-Left Corner (X: 390735, Y:6512015), Lower-Right Corner (X: 402075, Y:6505685), Pixel Size (X: 30m, Y: 30m); (b) ICTM DEM categorization; (c) ICTM 3D categorization; (d) ICTM status-limits categorization

that, in the sequential implementation, the data of all layers has to be stored in a unique processor. Table 3 presents the *speedups* for the tests realized. Observe that a parallel implementation may be considered when the `ICTM` presents more than one layer. However, it becomes really necessary in the case that it has a great amount of layers, since, in this case, a sequential implementation is practically not feasible.

Table 3. *Speedups* for the tests realized

Matrix	# of layers	e60-Eth	e800-Eth	e60-Myr	e800-Myr
M_1	3	0.97	1.45	0.97	1.61
M_1	5	1.48	2.18	1.46	2.63
M_2	3	1.43	2.66	1.34	2.57
M_2	5	2.37	4.36	2.25	4.27
M_3	3	18.91	34.41	17.37	32.81
M_3	5	41.83	76.43	37.99	72.74

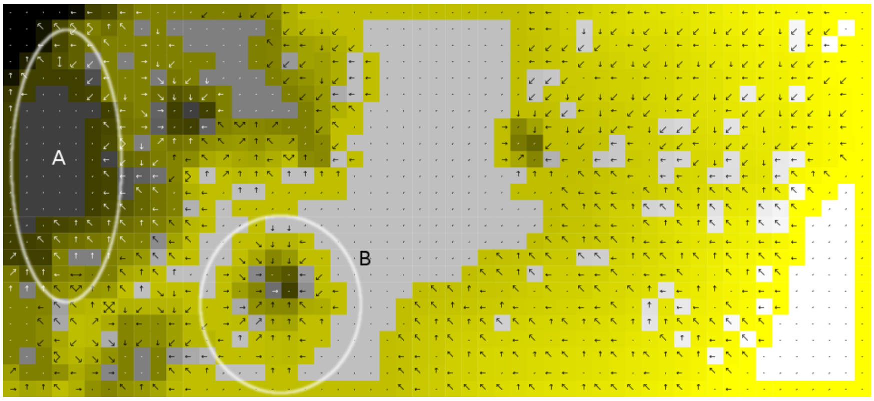


Fig. 5. Relief categorization of a portion of LANDSAT image with coordinates: Upper-left corner (X:427559m, Y:6637852m), Lower-right corner (X;480339m, Y:6614507m)

5 Discussion and Conclusion

In the categorizations produced by the ICTM, the state of a cell in relation to its neighbors, concerning the declivity, is shown directly by arrows (see Fig. 5), which has been considered a very intuitive representation, by the ecologists, since most geographic information systems present this kind of result by the usual color encoding of declivity, with no indication of direction.

The ICTM is regulated by two aspects, namely, the spacial resolution of the DEM and the neighborhood radius of the cell. Thus, regions with an agglomeration of limiting cells can be studied with more details by just increasing the resolution of altimetry data, or reducing the neighborhood radius. In plain areas (see Fig. 5 (region A)), a large neighborhood radius indicated reasonable approximations for the declivity degree. However, regions with too much declivity variation (see Fig. 5 (region B)) obtained good approximations only with small radius. The number of categories obtained is always inversely proportional to the neighborhood radius and to the area of a tessellation cell.

The analysis of some related works concerning image segmentation [3,6,8,10] turns out that those methods are, in general, heuristic, and, therefore, the ICTM model presented here is more reliable (for other works, see, e.g.: [2,11]).

Future work is concerned with the aggregation of a dynamic structure, based on cellular automata [12], for the modelling of the dynamic of populations in a ecological context.

Acknowledgments

We are very grateful to Alex Bager for his suggestions concerning ecological aspects. We also thank the referees for the valuable comments. This work is supported by the CNPq and FAPERGS. V.K. was also supported by NASA under cooperative agreement NCC5-209, by Future Aerospace Science and Technology Program Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under the grant F49620-00-1-0365), by NSF grants EAR-0112968 and EAR-0225670, by the Army Research Laboratories grant DATM-05-02-C-0046, and by IEEE/ACM SC2003 Minority Serving Institutions Participation Grant.

References

1. M.S. Aguiar, G.P. Dimuro, and A.C.R. Costa. `TOPO-ICTM`: an interval tessellation-based model for reliable topographic segmentation. *Numerical Algorithms* 37(1): 3–11, 2004.
2. D. Coblentz, V. Kreinovich, B. Penn, and S. Starks. Towards reliable sub-division of geological areas: interval approach. In L. Reznik and V. Kreinovich, editors, *Soft Computing in Measurements and Information Acquisition*, pages 223–233, 2003. Springer-Verlag.
3. M. C. Cooper. The Tractability of Segmentation and Scene Analysis. *International Journal on Computer Vision*, 30(1): 27–42, 1998.
4. R.B. Kearfort and V. Kreinovich (eds.). *Aplications of Interval Computations*. Kluwer, Dordrecht, 1996.
5. R.T.T. Forman. *Land Mosaics: the ecology of landscapes and regions*. Cambridge University Press, Cambridge, 1995.
6. K.S. Fu and J.K. Mui. A Survey on Image Segmentation. *Pattern Recognition*, 13(1): 3–16, 1981.
7. W. Gropp, E. Lusk e A. Skjellum. *Using MPI: portable Parallel Programming with the Message-Passing Interface*, MIT Press, 2nd ed., 1999.
8. J.L. Lisani, L. Moisan, P. Monasse, and J.M. Morel. On The Theory of Planar Shape. *Multiscale Modeling and Simulation*, 1(1): 1–24, 2003.
9. R.E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, 1979.
10. S.E. Umbaugh. *Computer Vision and Image Processing*. Prentice Hall, New Jersey, 1998.
11. K. Villaverde and V. Kreinovich. A Linear-Time Algorithm that Locates Local Extrenma of a Function of One Variable from Interval Measurements Results. *Interval Computations*, 4: 176–194, 1993.
12. S. Wolfram. *Cellular Automata and Complexity: selected papers*. Addison-Wesley, Readings, 1994.

Counting the Number of Connected Components of a Set and Its Application to Robotics

Nicolas Delanoue[1], Luc Jaulin[2], and Bertrand Cottenceau[1]

[1] Laboratoire d'Ingénierie des Systèmes Automatisés
LISA FRE 2656 CNRS, Université d'Angers
62, avenue Notre Dame du Lac - 49000 Angers
{nicolas.delanoue,bertrand.cottenceau}@istia.univ-angers.fr
[2] Laboratoire E^3I^2
ENSIETA, 2 rue François Verny
29806 Brest Cedex 09
luc.jaulin@ensieta.fr

Abstract. This paper gives a numerical algorithm able to compute the number of path-connected components of a set $\mathbb{S}$ defined by nonlinear inequalities. This algorithm uses interval analysis to create a graph which has the same number of connected components as $\mathbb{S}$. An example coming from robotics is presented to illustrate the interest of this algorithm for path-planning.

1 Introduction

There exist different kinds of algorithms for path-planning. Most of the approaches are based on the use of potential function introduced by Khatib [3]. This type of methods may be trapped in a local minimum and often fail to give any feasible path.

Interval analysis [7] is known to be able to give guaranteed results (See e.g. [2]). In the first section, the notion of feasible configuration space is recalled and it is shown why its topology can be a powerful tool for path-planning. In the next section, topological definitions and a sufficient condition to prove that a set is star-shaped are given. This sufficient condition is the key result of the `CIA` algorithm presented in the fourth section. This algorithm creates a graph which has the same number of connected components as $\mathbb{S}$ where $\mathbb{S}$ is a subset of $\mathbb{R}^n$ defined by non-linear inequalities. [9] and [8] give algorithms where $\mathbb{S}$ is (closed) semi-algebraic.

Throughout this article, we use a robot to illustrate a new path-planning algorithm.

2 Motivation with an Example Coming from Robotics

2.1 A Robot

Consider a 2-dimensional room which contains two walls (represented in gray in the Figure 1. The distance between the walls is y_0. A robotic arm with two degrees of freedom α and β is placed in this room. It is attached to a wall at a point O and has two links OA and AB.

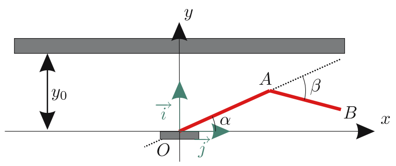


Fig. 1. A robotic arm with two links, $OA = 2$ and $AB = 1.5$

The Cartesian coordinates of A and B are given by the following equations:

$$\begin{cases} x_A = 2\cos(\alpha) \\ y_A = 2\sin(\alpha) \end{cases} \qquad \begin{cases} x_B = 2\cos(\alpha) + 1.5\cos(\alpha+\beta) \\ y_B = 2\sin(\alpha) + 1.5\sin(\alpha+\beta) \end{cases}$$

2.2 Configuration Set

Each coordinate of the *configuration space* represents a degree of freedom of the robot (See Figure 2). The number of independent parameters needed to specify an object configuration corresponds to the dimension of the configuration space. In our example, only α and β are necessary to locate the robot configuration, so our configuration space is a 2-dimensional space.

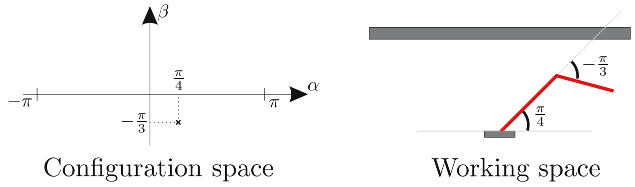


Fig. 2. A point in the configuration space (left) and its corresponding robot configuration

Since the robot cannot run through the walls, one has the following constraints $y_A \in [0, y_0]$ and $y_B \in]-\infty, y_0]$ and $\alpha \in [-\pi, \pi]$ and $\beta \in [-\pi, \pi]$. When these constraints are satisfied, the robot is said to be in a *feasible configuration*. The feasible configuration set $\mathbb{S}$ is thus defined as:

$$\mathbb{S} = \left\{ (\alpha, \beta) \in [-\pi, \pi]^2 / \begin{cases} 2\sin(\alpha) & \in [0, y_0] \\ 2\sin(\alpha) + 1.5\sin(\alpha+\beta) & \in]-\infty, y_0] \end{cases} \right\}$$

2.3 Connectedness of the Feasible Configuration Set and Path-Planning

Figure 3 shows how the feasible configuration set is affected by y_0. Three cases are presented:

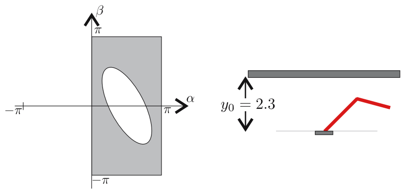


Fig 3.1

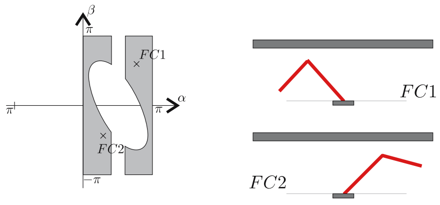


Fig 3.2

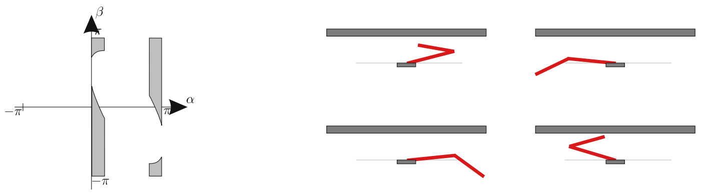


Fig 3.3

Fig. 3. - **Fig. 3.1.** Feasible configuration set when $y_0 = 2.3$. The robot can move from every initial feasible configuration to any goal feasible configuration. In this case, $\mathbb{S}$ has only one connected component. It is said *path-connected* (See Definition 1). - **Fig. 3.2.** Feasible configuration set when $y_0 = 1.9$. The configuration set has two path-connected components. It is impossible to move the robot from the first configuration $FC1$ to the second one $FC2$ without violating any constraint. - **Fig. 3.3.** Feasible configuration set when $y_0 = 1.1$. The robot can be trapped in four regions. $\mathbb{S}$ has four connected components. In each connected component, the robot can move but cannot reach any another components

In this article, a reliable method able to count the number of connected components of sets described by inequalities is presented. These sets can be feasible configuration sets. With a couple of configurations, we are able to guarantee that there exists or not a path to connect this ones. Moreover, when we have proven that two configurations are connectable, we are able to propose a path to connect them without violating any constraint.

3 Topological Brief Overview and a Key Result Leading to Discretization

In this section, definitions of a path-connected set and star-shaped set are recalled. Then, it is shown how this notions are linked. The last result is the key result leading to a robust discretization presented in the next section.

3.1 Topological Brief Overview

Definition 1. *A topological set* $\mathbb{S}$ *is* path-connected *if for every two points* $x, y \in \mathbb{S}$, *there is a continuous function* γ *from* $[0, 1]$ *to* $\mathbb{S}$ *such that* $\gamma(0) = x$ *and* $\gamma(1) = y$. *Path-connected sets are also called 0-connected.*

Definition 2. *A point v^* is a* star *for a subset X of an Euclidean set if X contains all the line segments connecting any of its points and v^*. A subset X of an Euclidean set is* star-shaped *or* v^*-star-shaped *if there exists $v^* \in X$ such that v^* is a star for X.*

Proposition 1. *A star-shaped set $\mathbb{S}$ is a path-connected set.*

Proof. Since $\mathbb{S}$ is star-shaped, there exists $v \in \mathbb{S}$ such that v is a star for $\mathbb{S}$. Let x and y be in $\mathbb{S}$ and:

$$\begin{aligned} \gamma : [0,1] &\to \mathbb{S} \\ t &\mapsto \begin{cases} (1-2t)x + 2tv & \text{if } t \in [0, \frac{1}{2}[\\ (2-2t)x + (2t-1)v & \text{if } t \in [\frac{1}{2}, 1]. \end{cases} \end{aligned}$$

γ is a continuous function from $[0,1]$ to $\mathbb{S}$ such that $\gamma(0) = x$ and $\gamma(1) = y$.

Proposition 2. *Let X and Y be two v^*-star-shaped set, then $X \cap Y$ is also v^*-star-shaped.*

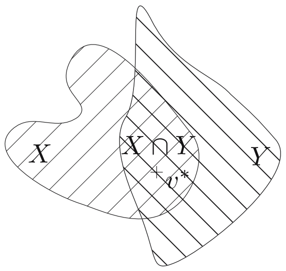


Fig. 4. Intersection stability

The next result is a sufficient condition to prove that a set defined by only one inequality is star-shaped. This sufficient condition can be checked using interval analysis (An algorithm such as SIVIA, Set Inversion Via Interval Analysis [5], can prove that equations (3.1) are inconsistent).

Proposition 3. *Let us define $\mathbb{S} = \{x \in D \subset \mathbb{R}^n | f(x) \leq 0\}$ where f is a C^1 function from D to $\mathbb{R}$, and D a convex set. Let v^* be in $\mathbb{S}$. If*

$$f(x) = 0,\ Df(x) \cdot (x - v^*) \leq 0,\ x \in D \tag{3.1}$$

is inconsistent then v^* is a star for $\mathbb{S}$.

Proof. See [1]

Remark 1. Combining this result with the Proposition 2, Proposition 3 can be used to prove that a set is star-shaped even if the set $\mathbb{S}$ is defined by several inequalities.

4 Discretization

The main idea of this disretization is to generate *star-spangled graph* which preserves the number of connected components of $\mathbb{S}$.

Definition 3. *A* star-spangled *graph of a set* $\mathbb{S}$*, noted by* $\mathcal{G}_{\mathbb{S}}$*, is a relation* $\mathcal{R}$ *on a paving*[3] $\mathcal{P} = \{p_i\}_{i\in I}$ *where:*

- *for all* p *of* $\mathcal{P}$, $\mathbb{S} \cap p$ *is star-shaped.*
- $\mathcal{R}$ *is the reflexive and symmetric relation on* $\mathcal{P}$ *defined by* $p\,\mathcal{R}\,q \Leftrightarrow \mathbb{S} \cap p \cap q \neq \emptyset$.
- $\mathbb{S} \subset \bigcup\limits_{i\in I} p_i$

Proposition 4. *Let* $\mathcal{G}_{\mathbb{S}}$ *be a star-spangled graph of a set* $\mathbb{S}$.
$\mathcal{G}_{\mathbb{S}}$ *has the same number of connected components as* $\mathbb{S}$. *i.e.* $\pi_0(\mathbb{S}) = \pi_0(\mathcal{G}_{\mathbb{S}})$[4].

Proof. See [1].

4.1 The Algorithm CIA

The algorithm called: `CIA` (path-**C**onnected using **I**nterval **A**nalysis) tries to generate a star-spangled graph $\mathcal{G}_{\mathbb{S}}$ (Proposition 4). The main idea is to test a suggested paving $\mathcal{P}$. In the case where the paving does not satisfy the condition that for all p in $\mathcal{P}$, $p \cap \mathbb{S}$ is star-shaped, the algorithm tries to improve this one by bisecting any boxes responsible for this failure.

For a paving $\mathcal{P}$, the algorithm checks for a box p of $\mathcal{P}$ whether $\mathbb{S} \cap p$ is star-shaped or not (Proposition 3.1 and 2), and to build its associated graph with the relation $\mathcal{R}$ mentioned before.

In Alg. 1 `CIA`[5], $\mathcal{P}_*, \mathcal{P}_{out}, \mathcal{P}_\Delta$ are three pavings such that $\mathcal{P}_* \cup \mathcal{P}_{out} \cup \mathcal{P}_\Delta = \mathcal{P}$, with $\mathcal{P}$ is a paving whose support is a (possibly very large) initial box X_0 (containing $\mathbb{S}$):

- the *star-spangled* paving $\mathcal{P}_*$ contains boxes p such that $\mathbb{S} \cap p$ is star-shaped.
- the *outer* paving $\mathcal{P}_{out}$ contains boxes p such that $\mathbb{S} \cap p$ is empty.
- the *uncertain* paving $\mathcal{P}_\Delta$, nothing is known about its boxes.

4.2 Application

Consider again the example presented in Section 1, the feasible configuration set $\mathbb{S}$ is:

$$\mathbb{S} = \left\{ (\alpha,\beta) \in [-\pi,\pi]^2 / \left\{ \begin{array}{ll} -2\sin(\alpha) & \leq 0 \\ 2\sin(\alpha) - y_0 & \leq 0 \\ 2\sin(\alpha) + 1.5\sin(\alpha+\beta) - y_0 & \leq 0 \end{array} \right. \right\} \tag{4.2}$$

When y_0 is equal to 2.3, 1.9 and 1.1, algorithm `CIA` generates these star-spangled graphs presented respectively on Figures 5,6 and 7.

[3] A paving is a finite collection of non overlapping n-boxes (Cartesian product of n intervals), $\mathcal{P} = \{p_i\}_{i\in I}$.

[4] In algebraic topology, $\pi_0(\mathbb{S})$ is the classical notation for the number of connected components of $\mathbb{S}$.

[5] This algorithm has been implemented (CIA.exe) and can be found at http://www.istia.univ-angers.fr/˜delanoue/

Algorithm 1 CIA - path-Connected using Interval Analysis

```
Require: $\mathbb{S}$ a subset of $\mathbb{R}^n$, $X_0$ a box of $\mathbb{R}^n$
 1: Initialization: $\mathcal{P}_* \leftarrow \emptyset$, $\mathcal{P}_\Delta \leftarrow \{X_0\}$, $\mathcal{P}_{out} \leftarrow \emptyset$
 2: while $\mathcal{P}_\Delta \neq \emptyset$ do
 3:    Pull the last element of $\mathcal{P}_\Delta$ into the box $p$
 4:    if "$\mathbb{S} \cap p$ is proven empty" then
 5:       Push $\{p\}$ into $\mathcal{P}_{out}$, Goto Step 2.
 6:    end if
 7:    if "$\mathbb{S} \cap p$ is proven star-shaped" and if we can guarantee $\forall p_* \in \mathcal{P}^*, p \cap p_* \cap \mathbb{S}$ is
       empty or not then
 8:       Push $\{p\}$ into $\mathcal{P}_*$, Goto Step 2.
 9:    end if
10:    $Bisect(p)$ and Push the two resulting boxes into $\mathcal{P}_\Delta$
11: end while
12: $n \leftarrow$ Number of connected components of $\mathcal{G}_\mathbb{S}$ (i.e. the relation $\mathcal{R}$ on $\mathcal{P}_*$).
13: return "$\mathbb{S}$ has $n$ path-connected components"
```

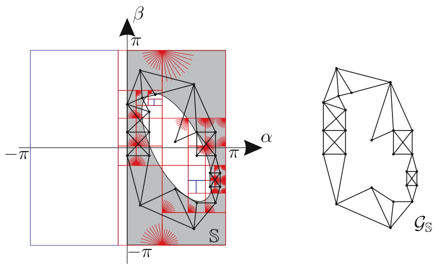


Fig. 5. The feasible configuration set and one of its star-spangled graph generated by CIA when $y_0 = 2.3$. The star-spangled graph $\mathcal{G}_\mathbb{S}$ is connected. By using Proposition 4, we deduce that from every couple of endpoints, it is possible to create a path to connect this ones. Subsection 4.3 shows how a path can be found

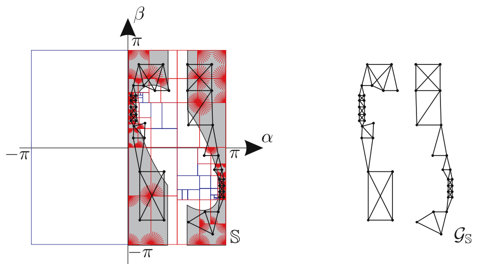


Fig. 6. The feasible configuration set and its star-spangled graph generated by CIA when $y_0 = 1.9$. Since $\mathcal{G}_\mathbb{S}$ has two connected components, we have a proof that $\mathbb{S}$ has two path-connected components

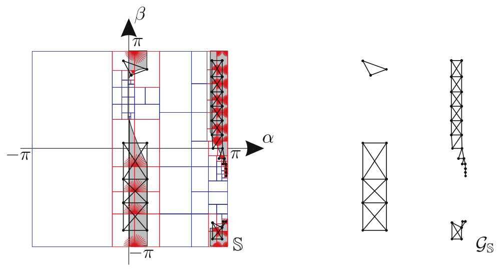


Fig. 7. The feasible configuration set and its star-spangled graph generated by `CIA` when $y_0 = 1.1$. $\mathcal{G}_{\mathbb{S}}$ and $\mathbb{S}$ have 4 connected components

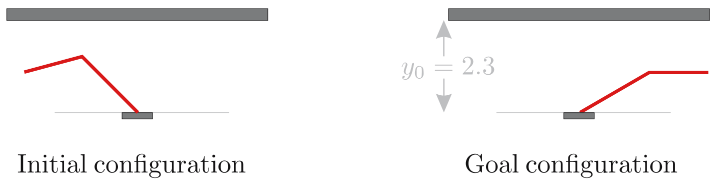


Fig. 8. Initial configuration, $x = (\frac{3\pi}{4}, \frac{\pi}{3})$ and goal configuration, $y = (\frac{\pi}{6}, -\frac{\pi}{6})$

4.3 Path-Planning

A star-spangled graph can be used to create a path between endpoints. Our goal is to find a path from the initial configuration x to the goal configuration y (e.g. Fig. 8).

As shown in Section 1, it suffices to find a path which connects x to y in the feasible configuration set. The algorithm `Path-planning with CIA`, thanks to a star-spangled graph, creates a path γ in $\mathbb{S}$. This algorithm uses the `Dijkstra` [6] algorithm which finds the shortest path between two vertices in a graph. Since $\mathcal{G}_{\mathbb{S}}$ is a star-shaped graph, every p in $\mathcal{P}$ is necessary star-shaped and we denote by v_p one of its stars.

Figure 9 shows the path γ created by `Path-planning with CIA`.

The corresponding configurations of the path γ are illustrated on Figure 10.

5 Conclusion

In this article, an algorithm which computes the number of connected components of a set defined by several non-linear inequalities has been presented. This dicretization makes possible to create a feasible path in $\mathbb{S}$ (Alg. 2). One of the main limitations of the proposed approach is that the computing time increases exponentially with respect to the dimension of $\mathbb{S}$. At the moment, we do not have a sufficient condition about f (Proposition 3) to ensure that algorithm `CIA` will terminate.

Algorithm 2 `Path-planning with CIA`

```
Require: A set $\mathbb{S}$, $x, y \in \mathbb{S}$, $\mathcal{G}_\mathbb{S}$ a star spangled graph of $\mathbb{S}$ (The relation $\mathcal{R}$ on the paving $\mathcal{P}$).
Ensure: $\gamma \subset \mathbb{S}$ a path whose endpoints are $x$ and $y$.
 1: Initialization: $\lambda \leftarrow \emptyset$
 2: for all $p \in \mathcal{P}$ do
 3:     if $x \in p$ then $p_x \leftarrow p$; if $y \in p$ then $p_y \leftarrow p$
 4: end for
 5: if Dijkstra$(\mathcal{G}_\mathbb{S}, p_x, p_y)$ = "Failure" then
 6:     Return "$x$ and $y$ are in two different path-connected components"
 7: else
 8:     $(p_k)_{1 \leq k \leq n} = (p_x, \dots, p_y) \leftarrow$ Dijkstra$(\mathcal{G}_\mathbb{S}, p_x, p_y)$
 9: end if
10: $\gamma \leftarrow [x, v_{p_x}]$
11: for $k \leftarrow 2$ to $n-1$ do
12:     $w_{k-1,k} \leftarrow$ a point in $p_{k-1} \cap p_k \cap \mathbb{S}$;   $w_{k,k+1} \leftarrow$ a point in $p_k \cap p_{k+1} \cap \mathbb{S}$
13:     $\gamma \leftarrow \gamma \cup [w_{(k-1,k)}, v_{p_k}] \cup [v_{p_k}, w_{(k,k+1)}]$
14: end for
15: $\gamma \leftarrow \gamma \cup [v_{p_y}, y]$
```

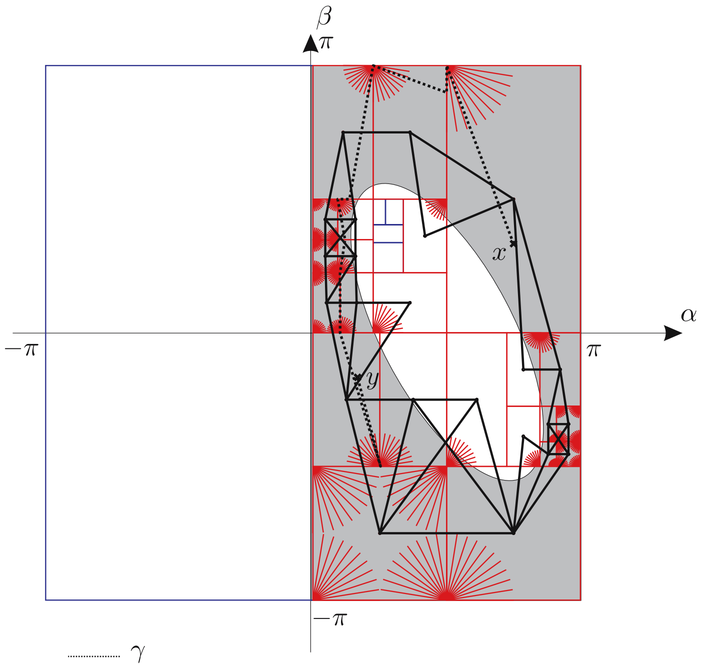


Fig. 9. Path γ generated by `Path-planning with CIA` from x to y when $y_0 = 2.3$

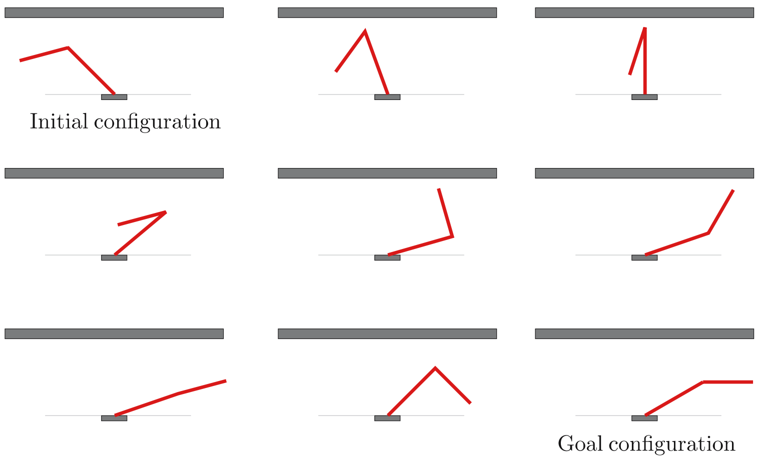


Fig. 10. Corresponding robot motion from the initial to the goal configuration

References

1. Delanoue, N., Jaulin,L., Cottenceau,B. Using interval arithmetic to prove that a set is path-connected. *Theoretical computer science, Special issue: Real Numbers and Computers.*, 2004.
2. L. Jaulin. Path planning using intervals and graphs. Reliable Computing, issue 1, volume 7, 2001
3. Khatib, O. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal Of Robotics Research*, 1986.
4. Janich, K. Topology (Undergraduate Texts in Mathematics) Springer Verlag
5. Jaulin, L. and Walter, E., Set inversion via interval analysis for nonlinear bounded- error estimation, Automatica, 29(4), 1993, 1053-1064.
6. Dijkstra, E.W.,. A note on two problems in connection with graphs, Numerische Math, 1, 1959, 269-271.
7. R. E. Moore, 1979, Methods and Applications of Interval Analysis SIAM, Philadelphia, PA
8. F. Rouillier, M.-F. Roy, M. Safey. Finding at least one point in each connected component of a real algebraic set defined by a single equation, Journal of Complexity 16 716-750 (2000)
9. S. Basu, R. Pollackz, M.-F. Roy. Computing the first Betti number and the connected components of semi-algebraic sets,
10. T. Lozano-Pérez, 1983, Spatial Planning: A Configuration Space Approach. IEEETC

Interval-Based Markov Decision Processes for Regulating Interactions Between Two Agents in Multi-agent Systems[⋆]

Graçaliz P. Dimuro[1] and Antônio C.R. Costa[1,2]

[1] Escola de Informática, Universidade Católica de Pelotas
Rua Felix da Cunha 412, 96010-000 Pelotas, Brazil
{liz,rocha}@ucpel.tche.br
[2] Programa de Pós-Graduação em Computação
Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves 9500, 91501-970 Porto Alegre, Brazil

Abstract. This work presents a model for Markov Decision Processes applied to the problem of keeping two agents in equilibrium with respect to the values they exchange when they interact. Interval mathematics is used to model the qualitative values involved in interactions. The optimal policy is constrained by the adopted model of social interactions. The MDP is assigned to a supervisor, that monitors the agents' actions and makes recommendations to keep them in equilibrium. The agents are autonomous and allowed to not follow the recommendations. Due to the qualitative nature of the exchange values, even when agents follow the recommendations, the decision process is non-trivial.

1 Introduction

There are many different techniques to deal with the problem of choosing optimal agent actions [11,13], some of them considering stochastic domains. The work presented in [3] deals with this problem using techniques from operations research, namely the theory of Markov Decision Processes (MDP) [5,8,12]. In this paper we introduce a qualitative version of a MDP, called *Qualitative Interval-based Markov Decision Process* (QI–MDP). The values characterizing the states and actions of the model are based on intervals and their calculation performed according to Interval Arithmetic [6]. The model is said to be qualitative in the sense that intervals are considered equivalent according to a loose equivalence relation. We apply the QI–MDP model to the analysis of the equilibrium of social exchanges between two interacting agents. The equilibrium is determined according to the balance of values the agents exchange during their interactions. The decision process pertains to a third agent, the *equilibrium supervisor*, who is in charge of giving recommendations to the agents on the best exchanges they can perform in order to keep the balance in equilibrium.

We modelled the social interactions according to Piaget's theory of exchange values [7], and derived the idea of casting the equilibrium problem in terms of a MDP from George Homans' approach to that same problem [4]. Due the lack of space, we shall

[⋆] This work was partially supported by CTINFO/CNPq and FAPERGS.

not consider in detail the social model based on Piaget's theory, since it was deeply explored in previous work [1,9,10]. A first application of this model in multi-agent systems was presented in [9,10]. In [1], exchange values were proposed for the modelling of collaborative on-line learning interactions.

The paper is organized as follows. The model of social interactions is presented in Sect. 2, and intervals are introduced in Sect. 3. The QI–MDP model is introduced in Sect. 4. Section 5 discusses the results. Section 6 is the Conclusion.

2 Social Reasoning About Exchange Values

The *Social Exchange Model* introduced by Piaget [7] is based on the idea that social relations can be reduced to social exchanges between individuals. Each social exchange is a service exchange between individuals and it is also concerned with an exchange of values between such individuals. The exchange values are of qualitative nature and are constrained by a *scale of values*.

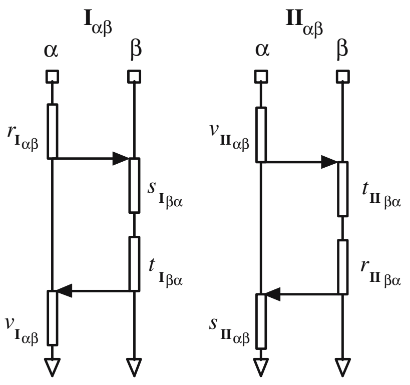
A *social exchange* is assumed to be performed in two stages. Figure 1 shows a schema of the exchange stages. In the stage $\mathrm{I}_{\alpha\beta}$, the agent α realizes a service for the agent β. The values related with this exchange stage are the following: *(i)* $r_{\mathrm{I}_{\alpha\beta}}$ is the value of the *investment*[3] done by α for the realization of a service for β; *(ii)* $s_{\mathrm{I}_{\beta\alpha}}$ is the value of β's *satisfaction* due to the receiving of the service done by α; *(iii)* $t_{\mathrm{I}_{\beta\alpha}}$ is the value of β's *debt*, the debt it acquired to α for its satisfaction with the service done by α; *(iv)* $v_{\mathrm{I}_{\alpha\beta}}$ is the value of the *credit* that α acquires from β for having realized the service for β. In the stage $\mathrm{II}_{\alpha\beta}$, the agent α asks the payment for the service previously done for the agent β, and the values related with this exchange stage have similar meaning. $r_{\mathrm{I}_{\alpha\beta}}$, $s_{\mathrm{I}_{\beta\alpha}}$, $r_{\mathrm{II}_{\beta\alpha}}$ and $s_{\mathrm{II}_{\alpha\beta}}$ are called *material values*. $t_{\mathrm{I}_{\beta\alpha}}$, $v_{\mathrm{I}_{\alpha\beta}}$, $t_{\mathrm{II}_{\beta\alpha}}$ and $v_{\mathrm{II}_{\alpha\beta}}$ are the *virtual values*. The order in which the exchange stage may occur is not necessarily $\mathrm{I}_{\alpha\beta} - \mathrm{II}_{\alpha\beta}$.

Piaget's approach to social exchange was an algebraic one: what interested him was the algebraic laws that define equilibrium of social exchanges. George Homans [4] approached the subject from a different view: he was interested in explaining how and why agents strive to achieve equilibrium in such exchanges. The solution he found, based on a behavioristic explanation of the agents' decision, suggested that agents look for a maximum of profit, in terms of social values, when interacting with each other. That proposal gave the starting point for the formalization we present below, where the looking for a maximum of profit is understood as a MDP to be solved by the equilibrium supervisor.

3 Modelling Social Exchanges with Interval Values

Piaget's concept of scale of values [7] is now interpreted in terms of Interval Mathematics [6]. Consider the set $\mathbb{IR}_L = \{[a,b] \mid -L \leq a \leq b \leq L, a, b \in \mathbb{R}\}$ of real intervals bounded by $L \in \mathbb{R}$ $(L > 0)$ and let $\mathcal{IR}_L = (\mathbb{IR}_L, +, \Theta, \widetilde{\ })$ be a *scale of interval exchange values*, where:

[3] An investment value is always negative.



$\mathbf{I}_{\alpha\beta}$
α
β
$r_{\mathbf{I}\alpha\beta}$
$s_{\mathbf{I}\beta\alpha}$
$t_{\mathbf{I}\beta\alpha}$
$v_{\mathbf{I}\alpha\beta}$
$\mathbf{II}_{\alpha\beta}$
α
β
$v_{\mathbf{II}\alpha\beta}$
$t_{\mathbf{II}\beta\alpha}$
$r_{\mathbf{II}\beta\alpha}$
$s_{\mathbf{II}\alpha\beta}$

Fig. 1. Stages of social exchanges

(i) $+ : \mathbb{IR}_L \times \mathbb{IR}_L \to \mathbb{IR}_L$ is the *addition* operation $[a, b] + [c, d] = [\max\{-L, a + c\}, \min\{b + d, L\}]$.
(ii) A *null value* is any $[a, b] \in \mathbb{IR}_L$ such that $mid([a, b]) = 0$, where $mid([a, b]) = \frac{a+b}{2}$ is the mid point of $[a, b]$. The set of null values is denoted by Θ. $[0, 0]$ is called the *absolute null value*.
(iii) A *quasi-symmetric value* for $X \in \mathbb{IR}_L$ is any $X' \in \mathbb{IR}_L$ such that $X + X' \in \Theta$. The set of quasi-symmetric values of X is denoted by $\widetilde{X}$.

$\mu\widetilde{X} \in \widetilde{X}$ is said to be the *least quasi-symmetric value* of X, if whenever there exists $S \in \widetilde{X}$ it holds that $d(\mu\widetilde{X}) \leq d(S)$, where $d([a, b]) = b - a$ is the diameter of an interval $[a, b]$. A *qualitative equivalence relation* $\approx$ is defined on $\mathbb{IR}_L$ by $X \approx Y \Leftrightarrow \exists Y' \in \widetilde{Y} : X + Y' \in \Theta$. For all $X \in \mathbb{IR}_L$, it follows that:

Proposition 1. *(i)* $\widetilde{X} = \{-[mid(X) - k, mid(X) + k] \mid k \in \mathbb{R} \wedge k \geq 0\}$*; (ii)* $\mu\widetilde{X} = -[mid(X), mid(X)]$.

Proof. $mid(X + (-[mid(X) - k, mid(X) + k])) = mid([\frac{a_1 - a_2 - 2k}{2}, \frac{a_2 - a_1 + 2k}{2}]) = 0$, for $X = [a_1, a_2]$. If $S \in \widetilde{X}$ is such that $mid(S) \neq mid(X)$, then $mid(X + S) = mid([\frac{a_1 - a_2 - 2k_2}{2}, \frac{a_2 - a_1 + 2k_1}{2}]) \neq 0$, for $k_1 \neq k_2 \in \mathbb{R}$, which is a contradiction. □

For practical applications, we introduce the concept of *absolute ϵ-null value* $0_\epsilon = [-\epsilon, +\varepsilon]$, with $\epsilon \in \mathbb{R}$ ($\epsilon \geq 0$) being a given tolerance. In this case, an ϵ-null value is any $N \in \mathbb{IR}_L$ such that $mid(N) \in 0_\epsilon$. The set of ϵ-null values is denoted by Θ_ϵ. The related set of ϵ-quasi-symmetric values of $X \in \mathbb{IR}_L$ is denoted by $\widetilde{X}_\epsilon$.

Let T be a set of discrete instants of time. Let α and β be any two agents. A *qualitative interval exchange-value system* for modelling the exchanges from α to β is a structure $\mathbf{IR}_{\alpha\beta} = (\mathcal{IR}_L; r_{\mathrm{I}_{\alpha\beta}}, r_{\mathrm{II}_{\beta\alpha}}, s_{\mathrm{I}_{\beta\alpha}}, s_{\mathrm{II}_{\alpha\beta}}, t_{\mathrm{I}_{\beta\alpha}}, t_{\mathrm{II}_{\beta\alpha}}, v_{\mathrm{I}_{\alpha\beta}}, v_{\mathrm{II}_{\alpha\beta}})$ where $r_{\mathrm{I}_{\alpha\beta}}, r_{\mathrm{II}_{\beta\alpha}} : T \to \mathbb{IR}_L$, $s_{\mathrm{II}_{\alpha\beta}}, s_{\mathrm{I}_{\beta\alpha}} : T \to \mathbb{IR}_L$, $t_{\mathrm{I}_{\beta\alpha}}, t_{\mathrm{II}_{\beta\alpha}} : T \to \mathbb{IR}_L$ and $v_{\mathrm{I}_{\alpha\beta}}, v_{\mathrm{II}_{\alpha\beta}} : T \to \mathbb{IR}_L$ are partial functions that evaluate, at each time instant $t \in T$, the investment, satisfaction, debt and credit values[4], respectively, involved in the exchange. Denote

[4] The values are undefined if no service is done at all at a given moment $t \in T$.

$r_{\mathrm{I}_{\alpha\beta}}(t) = r^t_{\mathrm{I}_{\alpha\beta}}$, $r_{\mathrm{II}_{\beta\alpha}}(t) = r^t_{\mathrm{II}_{\beta\alpha}}$, $s_{\mathrm{II}_{\alpha\beta}}(t) = s^t_{\mathrm{II}_{\alpha\beta}}$, $s_{\mathrm{I}_{\beta\alpha}}(t) = s^t_{\mathrm{I}_{\beta\alpha}}$, $t_{\mathrm{I}_{\beta\alpha}}(t) = t^t_{\mathrm{I}_{\beta\alpha}}$, $t_{\mathrm{II}_{\beta\alpha}}(t) = t^t_{\mathrm{II}_{\beta\alpha}}$, $v_{\mathrm{I}_{\alpha\beta}}(t) = v^t_{\mathrm{I}_{\alpha\beta}}$ and $v_{\mathrm{II}_{\alpha\beta}}(t) = v^t_{\mathrm{II}_{\alpha\beta}}$. A *configuration of exchange values* is specified by one of the tuples $(r^t_{\mathrm{I}_{\alpha\beta}}, s^t_{\mathrm{I}_{\beta\alpha}}, t^t_{\mathrm{I}_{\beta\alpha}}, v^t_{\mathrm{I}_{\alpha\beta}})$ or $(v^t_{\mathrm{II}_{\alpha\beta}}, t^t_{\mathrm{II}_{\beta\alpha}}, r^t_{\mathrm{II}_{\beta\alpha}}, s^t_{\mathrm{II}_{\alpha\beta}})$. The sets of configurations of exchange values from α to β, for stages I and II, are denoted by $\mathrm{EV}^{\mathrm{I}}_{\mathbf{IR}_{\alpha\beta}}$ and $\mathrm{EV}^{\mathrm{II}}_{\mathbf{IR}_{\alpha\beta}}$, respectively.

Consider the functions $\mathrm{I}_{\alpha\beta} : T \rightarrow \mathrm{EV}^{\mathrm{I}}_{\mathbf{IR}_{\alpha\beta}}$ and $\mathrm{II}_{\alpha\beta} : T \rightarrow \mathrm{EV}^{\mathrm{II}}_{\mathbf{IR}_{\alpha\beta}}$, defined, respectively, by $\mathrm{I}_{\alpha\beta}(t) = \mathrm{I}^t_{\alpha\beta} = (r^t_{\mathrm{I}_{\alpha\beta}}, s^t_{\mathrm{I}_{\beta\alpha}}, t^t_{\mathrm{I}_{\beta\alpha}}, v^t_{\mathrm{I}_{\alpha\beta}})$ and $\mathrm{II}_{\alpha\beta}(t) = \mathrm{II}^t_{\alpha\beta} = (v^t_{\mathrm{II}_{\alpha\beta}}, t^t_{\mathrm{II}_{\beta\alpha}}, r^t_{\mathrm{II}_{\beta\alpha}}, s^t_{\mathrm{II}_{\alpha\beta}})$. A *stage of social exchange* from α to β is either a value $\mathrm{I}^t_{\alpha\beta}$, where $r^t_{\mathrm{I}_{\alpha\beta}}$ is defined, or $\mathrm{II}^t_{\alpha\beta}$, where $r^t_{\mathrm{II}_{\beta\alpha}}$ is defined.

A *social exchange process* between any two agents α and β, occurring during the time instants $T = t_1, \ldots, t_n$, is any finite sequence $\mathrm{s}^T_{\{\alpha,\beta\}} = e_{t_1}, \ldots, e_{t_n}$, $n \geq 2$, of exchange stages from α to β and from β to α, where there are $t, t' \in T$, $t \neq t'$, with *well defined investment values* $r^t_{\mathrm{I}_{\alpha\beta}}$ and $r^{t'}_{\mathrm{II}_{\beta\alpha}}$ (or $r^t_{\mathrm{I}_{\beta\alpha}}$ and $r^{t'}_{\mathrm{II}_{\alpha\beta}}$).

The *material results* $M_{\alpha\beta}$ and $M_{\beta\alpha}$ of a social exchange process, from the points of view of α and β, respectively, are given by the respective sum of the material values involved in the process. Considering $k^T_{\mathrm{I}_{\lambda\delta}} = \sum_{t\in T} k^t_{\mathrm{I}_{\lambda\delta}}$ and $k^T_{\mathrm{II}_{\lambda\delta}} = \sum_{t\in T} k^t_{\mathrm{II}_{\lambda\delta}}$, for all well defined $k^t_{\mathrm{I}_{\lambda\delta}}$ and $k^t_{\mathrm{II}_{\lambda\delta}}$, with $k = r, s$, then $M_{\alpha\beta} = r^T_{\mathrm{I}_{\alpha\beta}} + s^T_{\mathrm{II}_{\alpha\beta}} + r^T_{\mathrm{II}_{\alpha\beta}} + s^T_{\mathrm{I}_{\alpha\beta}}$ and $M_{\beta\alpha} = r^T_{\mathrm{I}_{\beta\alpha}} + s^T_{\mathrm{II}_{\beta\alpha}} + r^T_{\mathrm{II}_{\beta\alpha}} + s^T_{\mathrm{I}_{\beta\alpha}}$. The process is said to be in equilibrium if $M_{\alpha\beta} \in \Theta_\epsilon$ and $M_{\beta\alpha} \in \Theta_\epsilon$. If a material result of a social exchange process is not in equilibrium, then any ϵ-quasi-symmetric of $M_{\alpha\beta}$ ($M_{\beta\alpha}$) is called a *compensation* value from α's (β's) point of view.

4 Solving the Equilibration Problem Using QI–MDP

4.1 The Basics of an QI–MDP

We conceive that, in the context of a social exchange process between two agents, a third agent, called *equilibrium supervisor*, analyzes the exchange process and makes suggestions of exchanges to the two agents in order to keep the material results of exchanges in equilibrium. To achieve that purpose, the equilibrium supervisor models the exchanges between the two agents as a MDP, where the *states* of the model represent "possible material results of the overall exchanges" and the *optimal policies* represent "sequences of *actions* that the equilibrium supervisor recommends that the interacting agents execute".

Consider $\epsilon, L \in \mathbb{R}$ ($\epsilon \geq 0, L > 0$), $n \in \mathbb{N}$ ($n > 0$) and let $\hat{E} = \{E^{-n}, \ldots, E^n\}$ be the set of equivalence classes of intervals, defined, for $i = -n, \ldots, n$, as:

$$E^i = \begin{cases} \{X \in \mathbb{IR}_L \mid i\frac{L}{n} \leq mid(X) < (i+1)\frac{L}{n}\} & \text{if } -n \leq i < -1 \\ \{X \in \mathbb{IR}_L \mid -\frac{L}{n} \leq mid(X) < -\epsilon\} & \text{if } i = -1 \\ \{X \in \mathbb{IR}_L \mid -\epsilon \leq mid(X) \leq +\epsilon\} & \text{if } i = 0 \\ \{X \in \mathbb{IR}_L \mid \epsilon < mid(X) \leq \frac{L}{n}\} & \text{if } i = 1 \\ \{X \in \mathbb{IR}_L \mid (i-1)\frac{L}{n} < mid(X) \leq i\frac{L}{n}\} & \text{if } 1 < i \leq n. \end{cases} \tag{4.1}$$

The classes E^i are the supervisor representations of classes of unfavorable ($i < 0$), equilibrated ($i = 0$) and favorable ($i > 0$) material results of exchange balances.

Table 1. Specification of compensation intervals

State	Compensation Interval C^i	State	Compensation Interval C^i
$E^i_{-n\leq i<-1}$	$[-\left(\frac{2i+1}{2}\frac{L}{n}\right)-\epsilon, -\left(\frac{2i+1}{2}\frac{L}{n}\right)+\epsilon]$	$E^i_{1<i\leq n}$	$[\frac{(1-2i)}{2}\frac{L}{n}-\epsilon, \frac{(1-2i)}{2}\frac{L}{n}+\epsilon]$
E^{-1}	$[\frac{1}{2}\left(\frac{L}{n}+\epsilon\right)-\epsilon, \frac{1}{2}\left(\frac{L}{n}+\epsilon\right)+\epsilon]$	E^1	$[-\frac{1}{2}\left(\frac{L}{n}+\epsilon\right)-\epsilon,$ $-\frac{1}{2}\left(\frac{L}{n}+\epsilon\right)+\epsilon]$
E^0	$[0,0]$		

Whenever it is understood from the context, we shall denote by E^- (or E^+) any class $E^{i<0}$ (or $E^{i>0}$). The *accuracy* of the equilibrium supervisor is given by $\kappa_n = \frac{L}{n}$. ϵ is the admissible tolerance for the equilibrium point. The range of the midpoints of the intervals that belong to a class E^i is called the *representative* of the class E^i, denoted $[E^i]$. In this paper, whenever it is clear from the context, we shall identify a class E^i with its representative.

The states of the QI–MDP model are pairs of classes (E^i_α, E^j_β), representing the material results of the social exchange process from the point of view of the agents α and β. The pair of classes (E^0_α, E^0_β) is a *terminal* state, representing that the system is in equilibrium.

The *actions* considered in the model are state transitions $(X^i, X^j) : \hat{E}\times\hat{E} \rightarrow \hat{E}\times\hat{E}$, with $i, j = -n, \ldots, n$, defined by $(X^i, X^j)(E^i_\alpha, E^j_\beta) = (E^{i'}_\alpha, E^{j'}_\beta)$ if $mid([E^i_\alpha] + X^i) \in E^{i'}_\alpha$ and $mid([E^j_\beta] + X^j) \in E^{j'}_\beta$, which occur by the addition, to the representatives of the classes E^i_α and E^j_β, of intervals X^i and X^j that should be of the following types: (i) the *absolute ϵ-null value* $0_\epsilon = [-\epsilon, +\epsilon]$; (ii) a *compensation interval*, which is the least quasi-symmetric, denoted by C^i, of a class representative E^i; (iii) a *go-forward-k-step interval*, which is an interval, denoted by F^i_k, that transforms a class E^i into $E^{(i+k)\neq 0}$, with $i \neq L$; (iv) a *go-backward-k-step interval*, which is an interval, denoted by B^i_{-k}, that transforms a class E^i into $E^{(i-k)\neq 0}$, with $i \neq -L$.

The set $\mathcal{C}$ of compensation intervals is shown in Table 1. The set $\mathcal{F}$ of go-forward intervals and their respective effects are partially presented in Table 2. The set of go-backward intervals, denoted by $\mathcal{B}$, can be specified analogously.

For example, for a state of type

$$(E^i_\alpha, E^j_\beta)_{-n\leq i<-1, 1<j\leq n} \equiv ([i\frac{L}{n}, (i+1)\frac{L}{n}], [(j-1)\frac{L}{n}, j\frac{L}{n}]),$$

the *compensation–compensation* action and the *go-backward*$_{-3}$–*go-forward*$_{+2}$ actions are given by (A1) $(C^i, C^j) = ([-\frac{2i+1}{2}\frac{L}{n}-\epsilon, -\frac{2i+1}{2}\frac{L}{n}+\epsilon], [\frac{(1-2j)}{2}\frac{L}{n}-\epsilon, \frac{(1-2j)}{2}\frac{L}{n}+\epsilon])$ and (A2) $(B^i_{-3}, F^j_{+2}) = ([-3\frac{L}{n}-\epsilon, -3\frac{L}{n}+\epsilon], [2\frac{L}{n}-\epsilon, 2\frac{L}{n}+\epsilon])$, respectively, resulting in the state transitions: $(E^i_\alpha, E^j_\beta)_{-n\leq i<-1, 1<j\leq n} \overset{(A1)}{\mapsto} (E^0_\alpha, E^0_\beta)$ and $(E^i_\alpha, E^j_\beta)_{-n\leq i<-1, 1<j\leq n} \overset{(A2)}{\mapsto} (E^{(i-3)}_\alpha, E^{(j+2)}_\beta)$.

The equilibrium supervisor has to find, for each state E, the action that shall achieve the terminal state or, at least, another state from where the terminal state can be achieved,

Table 2. Specification of some go-forward intervals and their respective effects

State	Go-forward interval F^i_{+k}	Effect
$E^i_{-n\leq i<-1}$	$\left[k\frac{L}{n}-\epsilon, k\frac{L}{n}+\epsilon\right]_{1-i\leq k\leq n-i-1}$	$E^i \mapsto E^{i+k}, 1 < i+k \leq n$
E^{-1}	$\left[k\frac{L}{n}-\epsilon, k\frac{L}{n}+2\epsilon\right]_{2<k\leq n}$	$E^{-1} \mapsto E^{-1+k}, 1 < -1+k \leq n$
E^0	$\left[k\frac{L}{n}, (k+1)\frac{L}{n}\right]_{0<k\leq n-1}$	$E^0 \mapsto E^{k+1}, 1 < k+1 \leq n$
E^1	$\left[k\frac{L}{n}-2\epsilon, k\frac{L}{n}+\epsilon\right]_{0<k\leq n-i}$	$E^1 \mapsto E^{1+k}, i < 1+k \leq n$
$E^i_{1<i\leq n}$	$\left[k\frac{L}{n}-\epsilon, k\frac{L}{n}+\epsilon\right]_{0<k\leq n-i}$	$E^i \mapsto E^{i+k}, i < i+k \leq n$

with the least number of steps. The choice of such actions is also regulated by the rules of the social exchanges, and, therefore, there are some state transitions that are not allowed. Based on a optimal policy, the equilibrium supervisor may be asked to recommend that the agents act optimally. An *optimal exchange recommendation* consists of a function that gives, for each actual material result (represented by a state of the model), a partially defined exchange stage that shall restore or establish the material equilibrium or, at least, give conditions that it be achieved in a least number of steps with least value uncertainty. The optimal exchange recommendation associates state transitions determined by the optimal policy with agents' social exchanges.

Although the interacting agents acknowledge the optimal recommendations from the equilibrium supervisor, they are autonomous in the sense that *they may not follow the recommendations exactly*. Thus, the system may achieve another state different from the one expected by the supervisor and, therefore, there may be a great deal of uncertainty about the effects of the agents actions. Even if the agents follow a recommendation exactly, we will show that the effect may not be the expected by the supervisor, since it depends on the ratio $\frac{\kappa_n}{\epsilon}$, where $\kappa_n = \frac{L}{n}$ is the equilibrium supervisor accuracy and ϵ $(0 \leq \epsilon < \kappa_n)$ is the admissible tolerance. On the other hand, we assume that there is never any uncertainty about the current state of the system, that is, the equilibrium supervisor always has access to the current configuration of exchange values and has complete and perfect abilities to evaluate the current material balance.

Definition 1. *A* Qualitative Interval Markov Decision Process *(QI–MDP) for keeping social exchanges in equilibrium is a tuple $\langle E, A, F, R\rangle^{L,n}_{\epsilon}$, where*[5]*:*

- The set of the states is the set of pairs of equivalence classes of intervals $E = E_\alpha \times E_\beta$, with $E_\lambda = \{E^i \mid i = -n, \ldots, -1, 0, 1, \ldots, n\}$ defined in (4.1).

- $A = (\mathcal{C} \cup \mathcal{F} \cup \mathcal{B} \cup \{[-\epsilon, +\epsilon]\}) \times (\mathcal{C} \cup \mathcal{F} \cup \mathcal{B} \cup \{[-\epsilon, +\epsilon]\})$ is the set of possible actions, where $\mathcal{C}$, $\mathcal{F}$ and $\mathcal{B}$ are the sets of compensation, go-forward and go-backward intervals, respectively.

- $F : E \times A \to \Pi(E)$ is the state-transition function, that gives for each state and each action, a probability distribution over the set of states;

- $R : (E \times A) \to \mathbb{R}$ is the reward function, giving the expected immediate reward gained by choosing an action a when the current state is e.

[5] In this model, the next state and the expected reward depend only on the previous state and the action taken, satisfying the so-called *Markov property*.

4.2 The Optimal Policy and the Reward Function

The reward function plays an important role when the equilibrium supervisor is choosing the action that will generate a recommendation of agents interaction, in each state. The supervisor aims to maximize the utility of sequences of actions, evaluated according to the reward function.

A sample reward function $R : (E \times A) \to \mathbb{R}$ that conforms to the idea of supporting a recommendation function that is able to direct agents into social equilibrium is partially sketched in Table 3. This particular function illustrates various requirements that should be satisfied by all reward functions of the model. Observe, for instance, that if the current state is of the type (E^-, E^+), then the best action to be chosen is a *compensation-compensation* action (C, C), which results in a state transition $(E^-, E^+) \mapsto (E^0, E^0)$. Any other choice will make the agents either take a long way to the equilibrium or get away from it.

On the other hand, if the current state is of type (E^-, E^-), then a *compensation-compensation* action (C, C) would generate a recommendation of agent exchanges of *satisfaction-satisfaction* type, which is impossible according to the model of social interactions [7], since it is impossible for an agent to get a satisfaction value from no service at all. The reward function R states that (C, C) is a very bad action to be chosen in such situation.

Any optimal policy $\pi^* : E \to A$ solving the social equilibrium problem should satisfy the set of requirements expressed by the schema partially sketched in Table 4 [6] The *optimal recommendation* associated to an optimal policy π^* is a function ρ_{π^*} that gives, for each state (E^i_α, E^j_β) and optimal action $\pi^*(E^i_\alpha, E^j_\beta) = (X^i, Y^j)$, a partial definition of a recommended exchange stage, consisting of pairs $((r_{\alpha\beta}, X^i), (s_{\beta\alpha}, Y^j))$ or $((r_{\beta\alpha}, Y^j), (s_{\alpha\beta}, X^i))$, where $(r_{\lambda\delta}, W)$ means the realization, by the agent λ, of a service with investment value $W < 0$, and $(s_{\delta\lambda}, W')$ means δ's satisfaction with interval value W', for receiving the service. The optimal recommendation ρ_{π^*} is also partially sketched in Table 4.

Table 3. Partial schema of the reward function R

R	(C,C)	$(0_\epsilon,C)$	$(C,0_\epsilon)$	(B_{-1},F_1)	(B_{-3},F_3)	(F_1,B_{-1})	(C,B_{-1})	(F_1,C)
(E^-,E^+)	30	20	-30	-5	-10	3	20	20
(E^+,E^+)	30	20	20	0	0	0	18	20
(E^-,E^-)	-30	-30	-30	30	0	30	28	-30

5 Discussion

In the following, consider that the *agents always follow the recommendations given by the equilibrium supervisor*. We show that, even in this favorable case, the decision process is a non-trivial one, due the qualitative nature of exchange values. The results concern the reachability of the terminal state show that under some conditions, it is

[6] Notice that it is a non deterministic policy.

Table 4. Partial schemata of the optimal policy π^* and associated optimal recommendation ρ_{π^*}

State	Optimal policy	Recommendation
$(E^i, E^j)^{-n\leq i<-1}_{1<j\leq n}$	$(C^i > 0, C^j < 0)$	$((r_{\beta\alpha}, C^j), (s_{\alpha\beta}, C^i))$
$(E^i, E^j)_{1<i,j\leq n}$	$(C^i < 0, C^j < 0)$	$((r_{\alpha\beta}, C^i), (s_{\beta\alpha}, C^j))$ or $((r_{\beta\alpha}, C^j), (s_{\alpha\beta}, C^i))$
$(E^0, E^j)_{1<j\leq n}$	$(0_\epsilon, C^j < 0)$	$((r_{\beta\alpha}, C^j), (s_{\alpha\beta}, 0_\epsilon))$
$(E^0, E^i)_{-n\leq i<-1}$	$(B^0_{-1} < 0, F^i_{+(-i+1)} > 0)$	$((r_{\alpha\beta}, B^0_{-1}), (s_{\beta\alpha}, F^i_{+(-i+1)}))$
$(E^1, E^i)_{-n\leq i<-1}$	$(B^1_{-1} < 0, C^i > 0)$	$((r_{\alpha\beta}, B^1_{-1}), (s_{\beta\alpha}, C^i))$
(E^{-1}, E^1)	$(F^{-1}_{+1} > 0, B^1_{-1} < 0)$	$((r_{\beta\alpha}, B^1_{-1}), (s_{\alpha\beta}, F^{-1}_{+1}))$
(E^1, E^{-1})	$(B^1_{-1} < 0, F^{-1}_{+1} > 0)$	$((r_{\alpha\beta}, B^1_{-1}), (s_{\beta\alpha}, F^{-1}_{+1}))$
$(E^i, E^1)_{-n\leq i<-1}$	$(C^i > 0, B^1_{-1} < 0)$	$((r_{\beta\alpha}, B^1_{-1}), (s_{\alpha\beta}, C^i))$
(E^{-1}, E^0)	$(F^{-1}_{+1} > 0, B^0_{-1} < 0)$	$((r_{\beta\alpha}, B^0_{-1}), (s_{\alpha\beta}, F^{-1}_{+1}))$
(E^0, E^{-1})	$(B^0_{-1} < 0, F^{-1}_{+1} > 0)$	$((r_{\alpha\beta}, B^0_{-1}), (s_{\beta\alpha}, F^{-1}_{+1}))$
$(E^i, E^j)_{-n\leq i,j<-1}$	$(F^i_{+(-i+1)} > 0, B^j_{-1} < 0)$ or $(B^j_{-1} < 0, F^i_{+(-i+1)} > 0)$	$((r_{\beta\alpha}, B^j_{-1}), (s_{\alpha\beta}, F^i_{+(-i+1)})$ or $((r_{\alpha\beta}, B^j_{-1}), (s_{\beta\alpha}, F^i_{+(-i+1)})$

always possible to have the system equilibrated in at most four steps. Let $M^\tau_{\alpha\beta}$ and $M^\tau_{\beta\alpha}$ be the material results of an exchange process, according to the points of view of the agents α and β, respectively, at step τ.

Proposition 2. *If $M^0_{\alpha\beta} \in E^{-1}$ and $M^0_{\beta\alpha} \in E^1$, then the system achieves the equilibrium in one step if and only if $1 < \frac{\kappa_n}{\epsilon} \leq 3$, $\epsilon > 0$.*

Proof. ($\Rightarrow$) If the system is at the state (E^{-1}, E^1), then, for the β's material result, it holds that $\epsilon < mid(M^0_{\beta\alpha}) \leq \frac{L}{n}$, and the optimal recommendation (Table 4, $row7$) is based on the optimal action $(C, C) = \left[-\frac{1}{2}\left(\frac{L}{n} + \epsilon\right), -\frac{1}{2}\left(\frac{L}{n} + \epsilon\right)\right]$. It follows that: $\epsilon - \frac{1}{2}\left(\frac{L}{n} + \epsilon\right) < mid(M^0_{\beta\alpha}) - \frac{1}{2}\left(\frac{L}{n} + \epsilon\right) \leq \frac{L}{n} - \frac{1}{2}\left(\frac{L}{n} + \epsilon\right) \Rightarrow \frac{1}{2}\left(-\frac{L}{n} + \epsilon\right) < mid(M^1_{\beta\alpha}) \leq \frac{1}{2}\left(\frac{L}{n} - \epsilon\right) \Rightarrow \frac{1}{2}\left(-h\epsilon + \epsilon\right) < mid(M^1_{\beta\alpha}) \leq \frac{1}{2}\left(h\epsilon - \epsilon\right)$, where $\frac{L}{n} = h\epsilon$, with $h > 1$. If the system achieves the equilibrium in the step 1, then it holds that $\frac{1}{2}\left(h\epsilon - \epsilon\right) \leq \epsilon$. It follows that $1 < h \leq 3$, and therefore, $1 < \frac{\kappa_n}{\epsilon} \leq 3$, since $\kappa_n = \frac{L}{n}$. The proofs for α's material result and of ($\Leftarrow$) are analogous. □

Proposition 3. *(i) If $M^0_{\alpha\beta} \in E^i$, with $1 < i \leq n$, then it is possible to get $M^\tau_{\alpha\beta} \in E^0_\alpha$ in at most $\tau = 2$ steps if and only if $1 < \frac{\kappa_n}{\epsilon} \leq 3$; (ii) If $M^0_{\beta\alpha} \in E^i$, with $-n \leq i < -1$, then it is possible to get $M^\tau_{\beta\alpha} \in E^0_\beta$ in at most $\tau = 2$ steps if and only if $1 < \frac{\kappa_n}{\epsilon} \leq 3$.*

Proof. (i)($\Rightarrow$) If $(i-1)\frac{L}{n} \leq mid(M^0_{\alpha\beta}) < i\frac{L}{n}$ and the optimal recommendation (Table 4, $row2$) is based on the optimal action $C = \left[\frac{(1-2i)}{2}\frac{L}{n}, \frac{(1-2i)}{2}\frac{L}{n}\right]$, then $(i-1)\frac{L}{n} + \frac{(1-2i)}{2}\frac{L}{n} < mid(M^0_{\beta\alpha}) + \frac{(1-2i)}{2}\frac{L}{n} \leq i\frac{L}{n} + \frac{(1-2i)}{2}\frac{L}{n}$, that is, $-\frac{1}{2}\frac{L}{n} < mid(M^1_{\beta\alpha}) \leq \frac{1}{2}$. It holds that $M^1_{\beta\alpha} \in E^1_\alpha$. From Prop. 2, it follows that with more one step we can get the desired result. The proofs of (i)($\Leftarrow$) and (ii) are analogous. □

From Prop. 3 it follows that an individual transition from a material result that belongs to a class E^i, with $1 < i \leq n$ or $-n \leq i < -1$, to the equilibrium can be done in at most two steps ($E^i \mapsto E^1($ or $E^{-1}) \mapsto E^0$). However, in any interaction between two agents, combined transitions departing from a state (E^i, E^j) or (E^j, E^i), with $1 < i \leq n$ and $-n \leq j < -1$, may result in a state different from (E^1, E^{-1}), (E^{-1}, E^1) or (E^0, E^0). We may have, for example, (E^{-1}, E^0), and, in this case, it will not be possible to get the equilibrium in one more step, since any *compensation* or *go-forward* action for α is not allowed without a correspondent β's service. The solution given by the optimal policy is then to have a transition to (E^1, E^{-1}) and then, finally, to reach (E^0, E^0). Thus, the overall process takes three steps.

The worst case is when the interaction presents material results that belong to the state (E^i, E^j), with $-n \leq i, j < -1$, since two simultaneous positive compensation actions (that would require a recommendation of satisfaction values for the two agents without any service at all) are not allowed. In this case, the optimal recommendation (Table 4) leads the agents to get the material equilibrium in at most four steps, by one of the following transitions: $(E^i, E^j)_{-n \leq i,j < -1} \overset{row12}{\mapsto} (E^1, E^j)_{-n \leq j < -1} \overset{row6}{\mapsto} (E^0, E^{-1}) \overset{row11}{\mapsto}$ $(E^{-1}, E^1) \overset{row7}{\mapsto} (E^0, E^0)$, or $(E^i, E^j)_{-n \leq i,j < -1} \overset{row13}{\mapsto} (E^j, E^1)_{-n \leq j < -1}$ $\overset{row9}{\mapsto} (E^{-1}, E^0) \overset{row10}{\mapsto} (E^1, E^{-1}) \overset{row8}{\mapsto} (E^0, E^0)$.

6 Conclusion

This paper introduced the QI–MDP version of the Markov Decision Process. The combination of interval-based modelling and qualitative approach to the comparison of values of the model made it well suited for solving the problem of keeping social exchanges in equilibrium. From the point of view of Jean Piaget's theory of social interactions, the QI–MDP means a sound way of making practical use of the INRC group of social exchanges that structure the social interactions and defines its equilibrium problem [1]. The QI–MDP model is general enough to be applied to other problems, besides the problem of keeping social interactions in equilibrium. It can also be applied to equilibrium problems of other kinds of systems, besides systems of social exchanges, if such systems have one single equilibrium state.

Future work will be concerned with the case of an equilibrium supervisor that is not able to determine the material balance of social exchange processes with complete reliability (i.e., it is not allowed to know all the exchange values of the two agents). In this case, a partially observable Markov decision process (POMDP) shall be considered (see, p.ex., [3]), since the equilibrium supervisor shall be able to make external observations (also probabilistic) to help him to decide about the recommendations.

References

1. A.C.R. Costa and G.P. Dimuro. The Case for Using Exchange Values in the Modelling of Collaborative Learning Interactions. In J. Mostow and P. Tedesco, eds, *Proceedings of Workshop 9 in the 7th International Conference on Intelligent Tutoring Systems, ITS 2004*, pages 19–24, Maceió, 2004.
2. M. d'Inverno and M. Luck. *Understanding Agent Systems.* Springer, Berlin, 2001.
3. L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and Acting in Partially Observabe Stochastic Domains. *Artificial Intelligence*, 101(1):99–134, 1998.

4. G.C. Homans. *Social Behavior - Its Elementary Forms*. Harcourt, Brace & World, New York, 1961.
5. R.A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, 1960.
6. R.E. Moore. *Methods and Applications of Interval Analysis*, SIAM, Philad., 1979.
7. J. Piaget. *Socialogical Studies*. Routlege, London, 1995.
8. M.L. Puterman. *Markov Decision Processes – Discrete Stochastic Dynamic Programming*. Wiley, New York, 1994.
9. M.R. Rodrigues, A.C.R. Costa, and R. Bordini. A System of Exchange Values to Support Social Interactions in Artificial Societes. In *Proceeding of the Second International Conference on Autonomous Agnets and Multiagents Systems, AAMAS 2003*, pages 81–88, Melbourne, Australia, 2003. ACM.
10. M.R. Rodrigues and A.C.R. Costa. Using Qualitative Exchange Values to Improve the Modelling of Social Interactions. In D. Hales, B. Edmonds, E. Norling, and J. Rouchier, eds, *Procedings of 4th Workshop on Agent Based Simulations*, n. 2927 in Lecture Notes in Computer Science, pages 57–72, Melbourne, Australia, 2003.
11. S. Russel and P. Norvig. *Artificial Intelligence, a Modern Approach*. Prentice Hall, Reading, 2003.
12. D.J. White. *Markov Decision Processes*. Wiley, New York, 2002.
13. M. Wooldridge. *An Introduction to Multi-Agent Systems.* Wiley, New York, 2002.

A Domain Theoretic Account of Euler's Method for Solving Initial Value Problems

Abbas Edalat and Dirk Pattinson

Department of Computing, Imperial College London, UK

Abstract. This paper presents a method of solving initial value problems using Euler's method, based on the domain of interval valued functions of a real variable. In contrast to other interval based techniques, the actual computation of enclosures to the solution is not based on the code list (term representation) of the vector field that defines the equation, but assumes instead that the vector field is approximated to an arbitrary degree of accuracy. By using approximations defined over rational or dyadic numbers, we obtain proper data types for approximating both the vector field and the solution. As a consequence, we can guarantee the speed of convergence also for an implementation of the method. Furthermore, we give estimates on the algebraic complexity for computing approximate solutions.

1 Introduction

We consider initial value problems given by a system of differential equations

$$\dot{y}_i = v_i(y), \quad y_i(0) = 0 \quad (i = 1, \dots, n) \tag{1}$$

where the vector field $v : [-K, K]^n \to [-M, M]^n$ is continuous in a rectangle containing the origin. Our goal is to compute a function $y = (y_1, \dots, y_n) : [0, a] \to \mathbb{R}^n$ which satisfies (1), up to any given degree of accuracy.

Standard numerical packages usually compute approximations to a solution with good precision, but there is no guarantee on the correctness of the computed values; indeed it is easy to find examples where they output inaccurate results [7]. Interval Analysis [13,14] provides a method for computing guaranteed upper and lower enclosures of the solution of initial value problems, see e.g. [2,3,8,9,11] and the references therein for a survey of current interval techniques.

In the approach of interval analysis based on the Euler method, real numbers are represented as intervals and outward rounding is applied if the result of an operation is not machine representable. For many practical applications, these methods produce good enclosures, but one has no control over widening of intervals, which can make the result unduly large. As a consequence, implementations of interval methods are not guaranteed to produce approximations which actually converge to the solution of the problem, or satisfy an *a priori* estimate on the actual convergence speed.

These questions are addressed in the present paper using the framework of domain theory [1,6]. Based on the domain of interval valued functions of a real variable, we construct enclosures of the solutions of an IVP with an a priori guaranteed width. Moreover, our construction gives rise to proper data types, which can be directly implemented on a digital computer. This allows us to guarantee the speed of convergence also for existing implementations.

Our new approach is based on a sequence of successively finer approximations to the vector field. Using these approximations, we obtain enclosures of the solution of the problem, which are then shown to converge to the solution. As the approximations of the vector field can be defined using rational (or dyadic) numbers, no loss of precision is incurred, and we can therefore guarantee the convergence speed also for an implementation of our techniques. These new techniques for the Euler method follow closely those for the Picard method as developed recently in [5].

The main contributions of the paper are (i) to show that we can compute arbitrary tight enclosures of the solution using approximations of the vector field, and (ii) to show that these computations can be carried out on data types, defined over the rational or dyadic numbers. Furthermore, we give an estimate on the speed of convergence to the solution and an estimate of the algebraic complexity of computing approximations for two different realisations of Euler's technique.

Plan of the paper: We recall basic notions from domain theory in Section 2, and introduce two realisations of Euler's technique in Section 3, which are shown to produce approximations to the solution of the problem. We then add approximations of the vector field that defines the IVP (Section 4), and give an estimate on the speed of convergence of our method. Section 5 shows how our techniques can actually be implemented on a digital computer and gives the promised estimates on the algebraic complexity. Finally, the last section puts our results into perspective with related research.

2 Preliminaries and Notation

First note that the continuity assumption on v entails that v attains its maximum, and we can therefore restrict the range of v to $[-M, M]^n$ without loss of generality. For the expression (1) to be well defined, we make the standard assumption $aM \leq K$.

Our investigations are based on the *interval domain* $(\mathbf{I}\mathbb{R}, \sqsubseteq)$ where $\mathbf{I}\mathbb{R} = \{[\underline{a}, \overline{a}] \mid \underline{a} \leq \overline{a} \text{ and } \underline{a}, \overline{a} \in \mathbb{R}\} \cup \{\mathbb{R}\}$ ordered by reverse inclusion, i.e. $\alpha \sqsubseteq \beta$ if $\beta \subseteq \alpha$. For a compact rectangle $R \subseteq \mathbb{R}$, the sub-domain of compact intervals $[\underline{a}, \overline{a}] \subseteq R$ is denoted by $\mathbf{I}R$ with inherited order relation.

Note that both $\mathbf{I}\mathbb{R}$ and $\mathbf{I}R$, for $R \subseteq \mathbb{R}$ a compact interval, are *directed complete*: For a directed set $D \subseteq \mathbf{I}\mathbb{R}$ of intervals, the least upper bound $\bigsqcup D$ always exists and is given by $\bigcap D$. In interval terms, suprema of a directed subset of $\mathbf{I}\mathbb{R}$ correspond to Moore's principle of nested convergence of [13].

The order on an arbitrary directed complete partial order (dcpo, for short) $(D, \sqsubseteq)$ induces a topology on D, the so called *Scott topology*: We call a set $O \subseteq D$ *open*, if

1. it is *upward closed*, i.e. $d \in O$ and $d \sqsubseteq e$ implies $e \in O$
2. it is *inaccessible by directed suprema*, i.e. if $A \subseteq D$ is directed and $\bigsqcup A \in O$, then $a \in O$ for some $a \in A$.

In the case of the interval domain $\mathbf{I}\mathbb{R}$, a base of the Scott topology is given by subsets of the form $\{\alpha \in \mathbf{I}\mathbb{R} \mid \alpha \subseteq \beta^\circ\}$ for any $\beta \in \mathbf{I}\mathbb{R}$, where β° is the interior of β. It can easily be seen that the Scott topology is T_0 and convergence in the Scott topology implies convergence in the metric topology, used by Moore [14], but not vice versa. In the sequel of the paper, we always consider a dcpo, or a space of intervals, as equipped with the Scott topology.

Given an arbitrary set X, every function $f : X \to \mathbf{I}\mathbb{R}$ can be represented by a pair $(\underline{f}, \overline{f})$ representing the upper and the lower interval boundary of f, that is, $f(x) = [\underline{f}(x), \overline{f}(x)]$ for all $x \in X$. We write this as $f = [\underline{f}, \overline{f}]$. We often make use of the following crucial fact [1]:

Fact 1. *Suppose $f = [\underline{f}, \overline{f}] : \mathbb{R} \to \mathbf{I}\mathbb{R}$. Then f is Scott continuous iff $\underline{f}$ is lower and $\overline{f}$ is upper semi continuous.*

If the domain of a function is also a dcpo (topologised with the Scott topology), we have the following alternative characterisation of continuity:

Fact 2. *Suppose $(D, \sqsubseteq)$ and $(E, \sqsubseteq)$ are dcpos. Then a monotone function $f : D \to E$ is continuous iff $\bigsqcup_{a \in A} f(a) = f(\bigsqcup A)$ for all directed $A \subseteq D$.*

We also note that the space $X \Rightarrow D$ of continuous functions of type $X \to D$, for a topological space X and a dcpo D, is again a dcpo in the pointwise order: given $f, g : X \to D$, we put $f \sqsubseteq g$ if $f(x) \sqsubseteq g(x)$ for all $x \in X$. Hence we can view the space of continuous functions $X \to D$ as a topological space w.r.t. the Scott topology on $X \Rightarrow D$. In case $X = \{1, \dots, n\}$ with the discrete topology, we write D^n for $X \Rightarrow D$ and obtain the n-fold cartesian product of the dcpo D with itself. In the special case $D \subseteq \mathbf{I}\mathbb{R}$ is a sub-dcpo, D^n is canonically isomorphic to the dcpo of n-dimensional compact rectangles, and we will use this isomorphism without further mention.

For our purposes, the following spaces of functions are of particular interest:

1. The space $\mathcal{S} = [0, a] \Rightarrow \mathbf{I}[-K, K]^n$ (with the Euclidean topology on $[0, a]$) for constructing solutions of (1)
2. The space $\mathcal{V} = \mathbf{I}[-K, K]^n \Rightarrow \mathbf{I}[-M, M]^n$ of interval vector fields.

We use the notion of *width* to measure the quality of an approximation. Given $\alpha = ([\underline{a}_0, \overline{a}_0], \dots [\underline{a}_n, \overline{a}_n]) \in \mathbf{I}\mathbb{R}^n$, we put $w(\alpha) = \max\{\overline{a}_i - \underline{a}_i \mid 1 \le i \le n\}$, and for a function $f : X \to \mathbf{I}\mathbb{R}^n$ we let $w(f) = \sup\{w(f(x)) \mid x \in X\}$ and call f *real valued* if $w(f) = 0$ and identify f with the induced function $X \to \mathbb{R}^n$.

The relation between vector fields in the classical sense and interval vector fields is given by the notion of *extension*: we say that $u \in \mathcal{V}$ *extends* $v : [-K, K]^n \to [-M, M]^n$ if $u(\{x\}) = \{v(x)\}$ for all $x \in [-K, K]^n$. In the sequel, we assume that $u \in \mathcal{V}$ is an extension of the classical vector field v. Note that every continuous $v = (v_1, \dots, v_n) : [-K, K]^n \to [-M, M]$ has an extension, the *canonical extension* can_v whose i-th component is given by $\mathbf{I}[-K, K]^n \ni \alpha \mapsto \{v_i(x) \mid x \in \alpha\}$. We emphasise that our framework does not force us to work with the canonical extension of the classical vector field v.

Finally, we introduce integrals of interval valued functions, which we use in the construction of solutions of the IVP. Suppose $p \le q$ and $f : [p, q] \to \mathbf{I}\mathbb{R}$. Then the integral of $f = [\underline{f}, \overline{f}]$ is defined as $\int_p^q f(t)dt = [\int_p^q \underline{f}(t)dt, \int_p^q \overline{f}(t)dt]$. The existence of the integrals follows from lower (resp. upper) semi continuity of $\underline{f}$ (resp. $\overline{f}$). If $f = (f_1, \dots, f_n) : [p, q] \to \mathbf{I}\mathbb{R}^n$, we let $\int_p^q f(t)dt = (\int_p^q f_1(t)dt, \dots, \int_p^q f_n(t)dt)$. The following property follows easily from the monotone convergence theorem:

Fact 3. *The integration operator $\int_p^q : ([p, q] \Rightarrow \mathbf{I}\mathbb{R}^n) \to ([p, q] \Rightarrow \mathbf{I}\mathbb{R}^n)$, defined by $f \mapsto \lambda x. \int_p^x f(t)dt$, is monotone and continuous.*

3 Euler's Operator in Domain Theory

We use a formulation of Euler's operator similar to the one given by Moore. The results of this section are in essence standard [14] and are reproduced here in the framework of domain theory for the reader's convenience.

The formalisation of Euler's method for solving initial value problems relies on the notion of *partitions* of the interval $[0, a]$:

Definition 1 (Partitions).

1. *A* partition *of* $[0, a]$ *is a finite sequence* $(q_0, \dots, q_k)$ *of real numbers* $0 = q_0 < \dots < q_k = a$*; the set of partitions of* $[0, a]$ *is denoted by* $\mathcal{P}$.
2. *The* norm $|Q|$ *of a partition* $Q = (q_0, \dots, q_k)$ *is given by* $|Q| = \max_{1 \le i \le k} q_i - q_{i-1}$ *and its* minimal width *is* $m(Q) = \min_{1 \le i \le k} q_i - q_{i-1}$. *We denote the ratio between maximal and minimal width by* $r(Q) = |Q|/m(Q)$.
3. *A partition* $Q = (q_0, \dots, q_k)$ refines *a partition* $P = (p_0, \dots, p_l)$ *if* $\{p_0, \dots, p_l\} \subseteq \{q_0, \dots, q_k\}$*; this is denoted by* $P \sqsubseteq Q$.

We now introduce two different realisations of Euler's technique for solving IVPs. The first has better convergence properties whereas computing with the second turns out to be more efficient.

For the remainder of the paper, we fix an extension $u : \mathbf{I}[-K, K]^n \to \mathbf{I}[-M, M]^n$ of the classical vector field v. If $\alpha = ([\underline{a}_1, \overline{a}_1], \dots, [\underline{a}_n, \overline{a}_n]) \in \mathbf{I}\mathbb{R}^n$ and $r \in \mathbb{R}$, we write $\alpha \oplus r = ([\underline{a}_1 - r, \overline{a}_1 + r], \dots, [\underline{a}_n - r, \overline{a}_n + r])$ for the symmetric expansion of the interval vector α with the real constant r.

Definition 2. *Suppose* $Q = (q_0, \dots, q_n) \in \mathcal{P}$. *Then the* Euler operator with linear expansion $E^l : \mathcal{P} \times \mathcal{V} \to [0, a] \Rightarrow \mathbf{I}[-K, K]^n$ *is defined by*

$$E^l_u(Q)(x) = \begin{cases} (0, \dots, 0) & x = 0 \\ E^l_u(Q)(q_i) + \int_{q_i}^x u(E^l_u(Q)(q_i) \oplus (x - q_i)M)dt & q_i \le x \le q_{i+1} \end{cases}$$

for $x \in [0, a]$. *The* Euler operator with constant expansion *is given similarly by*

$$E^c_u(Q)(x) = \begin{cases} (0, \dots, 0) & x = 0 \\ E^c_u(Q)(q_i) + \int_{q_i}^x u(E^c_u(Q)(q_i) \oplus \Delta q_i M)dt & q_i \le x \le q_{i+1} \end{cases}$$

where $\Delta q_i = q_{i+1} - q_i$. *In the sequel,* E *stands for either* E^l *or* E^c

The operator with constant expansion represents an interval version of Euler's method for constructing solutions of differential equations, as described by Moore [14]. An equivalent definition could also be given without the use of integration. However, our definition allows us to treat both operators in the same framework, and therefore enables us to use the same proof techniques for both.

We collect some basic facts on the operators E^c and E^l. First, note that E_u is (i.e. E_u^l and E_u^c are) well defined, monotone and computes enclosures of the solution.

Proposition 4. *E_u is well defined, monotone ($E_u(Q) \in \mathcal{S}$ and $E_u(P) \sqsubseteq E_u(Q)$ whenever $P \sqsubseteq Q$) and satisfies $E_u(Q) \sqsubseteq z$ for any solution z of (1).*

Using the fact that every initial value problem of the form (1) has at least one solution, it is easy to see that if the supremum of the Euler iterates is real valued, it solves (1).

Corollary 5. *Suppose $\mathcal{P}$ is a directed set of partitions and $y = [\underline{y}, \overline{y}] = \bigsqcup_{Q \in \mathcal{P}} E_u(Q)$ and $\underline{y} = \overline{y}$. Then $\underline{y} = \overline{y}$ is a solution of (1).*

In order to be able to compute arbitrarily tight enclosures of the solution, we need to impose a Lipschitz condition on the vector field; this is as in the classical theory. The following definition translates this into an interval setting:

Definition 3 (Interval Lipschitz Condition). *The function $u : \mathbf{I}[-K, K]^n \to \mathbf{I}[-M, M]^n$ satisfies an* interval Lipschitz condition *with* Lipschitz constant *L if $w(u(\alpha)) \leq L \cdot w(\alpha)$ for all $\alpha \in \mathbf{I}[-K, K]^n$.*

For the rest of the paper, we assume that u is an extension of v, which satisfies an interval Lipschitz condition with Lipschitz constant L. The assumption that u is interval Lipschitz is actually equivalent to v satisfying a Lipschitz condition [5], hence our assumption is in accordance with the classical theory.

Assuming a Lipschitz condition, we can give guarantees on the speed of convergence. We begin with an auxiliary lemma which helps to show that in this case, the approximations converge to a real valued function. In particular, this lemma also shows that E^l has better convergence properties than E^c.

Lemma 6. *Suppose $Q = (q_0, \ldots, q_n)$ is a partition. Then*

$$w(E_u^*(Q)(x)) \leq w(E_u^*(Q)(q_i))(1 + |Q| \cdot L) + C|Q|^2 LM$$

for $x \in [q_i, q_{i+1}]$, where $C = 1$ for $* = l$ and $C = 2$ for $* = c$.

Based on the previous lemma, we can now give an estimate on the speed of convergence; recall from Definition 1 that $r(Q)$ is the ratio of the largest and smallest distance between two partition points.

Proposition 7. *Suppose P is a partition of $[0, a]$. Then*

$$w(E_u^*(P)) \leq C \cdot |Q| M (e^{aLr(Q)} - 1)$$

where $C = 1$ for $* = l$ and $C = 2$ for $* = c$.

By suitably modifying non-equidistant partitions, the term $r(Q)$ can be eliminated.

Corollary 8 (Speed of Convergence).

1. *If Q is equidistant, then $w(E_u^*(Q)) \leq C \cdot |Q|M(e^{aL} - 1)$.*
2. *If Q is arbitrary, then $w(E_u^*(Q)) \leq 2C|Q|M \cdot (e^{4aL} - 1)$.*

where, in both cases, $C = 1$ for $* = l$ and $C = 2$ for $* = c$.

Our main result is thus:

Theorem 9. *Suppose $(Q_n)_{n\in\mathbb{N}}$ is an increasing sequence of partitions with $\lim_{n\to\infty} |Q_n| = 0$ and $y = \bigsqcup_{n\in\mathbb{N}} E_u(Q_n)$. Then $w(y_n) \leq C \cdot |Q_n|$ for some $C \geq 0$ and $\bigsqcup_{n\in\mathbb{N}} y_n$ is real valued and a solution of (1).*

4 Approximation of the Vector Field

We have seen in the previous section how to construct approximations for the solution of an IVP *directly* in terms of the interval extension of the classical vector field itself. From a computational point of view, this is unrealistic. In practice, only approximations to the vector field up to an arbitrary degree of accuracy are available for computation. In this section, we show that Euler's operator E_u is continuous in u, which will allow us to use approximations of the vector field for computing the solution of the IVP up to an arbitrary degree of accuracy. Instead of assuming that the vector field is given as a term involving certain basic functions like arithmetic operations and trigonometric functions, we assume that the vector field is given as a supremum of simple functions, each of which takes only finitely many values. As we will see in the next section, the use of simple functions allows us to compute the solution without loss of accuracy, and we can therefore guarantee the convergence also for an implementation of the method. We follow the convention of the previous section and use E_u to stand for both E_u^l and E_u^c.

Lemma 10. *Suppose $u_1, u_2 : \mathbf{I}[-K, K]^n \to \mathbf{I}[-M, M]^n$ with $u_1 \sqsubseteq u_2$. Then $E_{u_1} \sqsubseteq E_{u_2}$, i.e. $E_{u_1}(Q) \sqsubseteq E_{u_2}(Q)$ for all partitions Q.*

Informally speaking, if u_1 contains more information than u_2, the operator associated with u_2 produces a better enclosure of the solution than that of u_1. We show that E_u is actually continuous in u, allowing us to use approximations of u for computing approximations of the solution.

Proposition 11. *Suppose $(u_j)_{j\in J}$ is a directed collection of vector fields u_j : $\mathbf{I}[-K, K]^n \to \mathbf{I}[-M, M]^n$ with $u = \bigsqcup_{j\in J} u_j$. Then $E_u = \bigsqcup_{j\in J} E_{u_j}$.*

As an immediate consequence, we deduce that continuity in u allows us to use approximations of u for computing solutions.

Corollary 12. *Suppose $(u_n)_{n\in\mathbb{N}}$ is a sequence in $\mathcal{V}$ with $u = \bigsqcup_{n\in\mathbb{N}} u_n$ and $(Q_n)_{n\in\mathbb{N}}$ is a sequence of partitions with $\lim_{n\to\infty} |Q_n| = 0$. Then $\bigsqcup_{n\in\mathbb{N}} E_{u_n}(Q_n)$ is real valued and satisfies the IVP (1).*

In presence of approximations u_n of u, the speed of convergence will clearly depend on the speed of convergence of the sequence u_n to u. We now introduce the measure which we use to express the convergence rate of u_n to u.

Definition 4. *If $\alpha = (\alpha_1, \dots, \alpha_n)$ and $\beta = (\beta_1, \dots, \beta_n) \in \mathbf{I}\mathbb{R}^n$, we let $d(\alpha, \beta) = \max\{|\underline{a}_i - \underline{b}_i|, |\overline{a}_i - \overline{b}_i|, i = 1, \dots, n\}$ where $\alpha_i = [\underline{a}_i, \overline{a}_i]$ and $\beta_i = [\underline{b}_i, \overline{b}_i]$. For $u, u' \in \mathcal{V}$, we put $d(u, u') = \sup_{\alpha \in \mathbf{I}[-K,K]^n} d(u(\alpha), u'(\alpha))$.*

Note that $d(\alpha, \beta)$ is the Hausdorff distance for compact intervals, already used by Moore [14]. The following Lemma is the key for obtaining a result on the speed of convergence in presence of approximations of the vector field.

Lemma 13. *Suppose $u' \sqsubseteq u$ and $Q = (q_0, \dots, q_k) \in \mathcal{P}$. Then*

$$w(E^*_{u'}(Q)(x)) \leq w(E^*_{u'}(Q)(p_i))(1 + |Q| \cdot L) + C|Q|^2 LM + |Q| d(u, u')$$

for $x \in [q_i, q_{i+1}]$, where $C = 1$ for $* = l$ and $C = 2$ for $* = c$.

Similar to the development in the previous section, we obtain the following global estimate.

Proposition 14. *Suppose Q is a partition of $[0, a]$ and $u' \sqsubseteq u$. Then*

$$w(E^*_{u'}(Q)) \leq C \cdot (|Q|M + \frac{d(u, u')}{L})(e^{aLr(Q)} - 1)$$

where $C = 1$ for $* = l$ and $C = 2$ for $* = c$.

Modifying the partitions which are used to obtain the above estimate, we can eliminate the term $r(Q)$ and obtain the following global estimate.

Corollary 15 (Speed of Convergence).

1. *If Q is equidistant, then $w(E^*_{u'}(Q)) \leq C \cdot (|Q|M + \frac{d(u,u')}{L})(e^{aL} - 1)$.*
2. *If Q is arbitrary, then $w(E^*_{u'}(Q)) \leq C \cdot (2|Q|M + \frac{d(u,u')}{L}) \cdot (e^{4aL} - 1)$.*

where, in both cases, $C = 1$ for $* = l$ and $C = 2$ for $* = c$.

In summary, we see that adding approximations to the vector field does not destroy the order of convergence speed, given that the approximations of the vector field converge as fast as the partitions decrease in width.

Theorem 16. *Suppose $(Q_n)_{n \in \mathbb{N}}$ is a monotone sequence of partitions of $[0, a]$ with $\lim_{n \to \infty} |Q_n| = 0$, $u = \bigsqcup_{n \in \mathbb{N}} u_n$ with $d(u, u_n) \leq C_0 \cdot |Q_n|$ for some constant $C_0 \geq 0$ and $y_n = E_{u_n}(Q_n)$. Then $w(y_n) \leq C_1 \cdot |Q_n|$ for some $C_1 \geq 0$ and $\bigsqcup_{n \in \mathbb{N}} y_n$ is real valued and solves the IVP (1).*

The next section shows, how we can implement the proposed method as to guarantee the speed of convergence also for actual implementations of the method.

5 Implementation of the Domain Theoretic Method

In this section, we demonstrate how the domain theoretic approach to solving initial value problems can be implemented on a digital computer in such a way that the estimates on the speed of convergence can be guaranteed for an implementation. The key concept here is that of a *base*. Informally speaking, a base of a directed complete partial D order is a collection $\mathcal{B} \subseteq D$ of elements which generate all of D by means of directed suprema. For the interval domain, it is easy to see that the intervals with rational (or dyadic) endpoints for a base, and we introduce suitable bases for the spaces $\mathcal{V}$ and $\mathcal{S}$ later. The main point about these bases is that (i) base elements form a proper data type and (ii) can be manipulated without any loss of precision.

The main contribution of this section is the proof that, if u is approximated by base elements, $E_u(Q)$ is also an element of the corresponding base. Furthermore, we give estimates on the algebraic complexity of computing $E_u(Q)$ both for E_u^c and E_u^l.

We refer the reader to [1, Section 2.2.2] for the formal definition of a base, and instead introduce the bases we work with in the sequel.

Definition 5. *Let $D \subseteq \mathbb{R}$ be a dense subset with $0, a \in D$ and assume that $0 = a_0 < \cdots < a_k = a$ with $a_0, \ldots, a_k \in D$, $\beta_0, \ldots, \beta_k \in \mathbf{I}[-K, K]^n_D$ and $\gamma_1, \ldots, \gamma_k \in \mathbf{I}[-M, M]^n_D$, where R_D denotes the set of rectangles, which are contained in R and whose endpoints lie in D^n. We write $\overline{\beta}_i$ for the vector representing the upper endpoints of the interval vector β_i with $\underline{\beta}_i$ given similarly. We consider the following classes of functions, where β^o is the interior of β:*

1. *The class $\mathcal{S}_D$ of piecewise D-linear functions $[0, a] \to \mathbf{I}[-K, K]^n$,*
$$f = (a_0, \ldots, a_k) \searrow (\beta_0, \ldots, \beta_k)$$
where $\overline{f}(x) = \overline{\beta}_{j-1} + \frac{x - a_{j-1}}{a_j - a_{j-1}}(\overline{\beta}_j - \overline{\beta}_{j-1})$ and $\underline{f}(x) = \underline{\beta}_{j-1} + \frac{x - a_{j-1}}{a_j - a_{j-1}}(\underline{\beta}_j - \underline{\beta}_{j-1})$ for $x \in [a_{j-1}, a_j]$. Every component of a D-linear function is piecewise linear and takes values in D at $a_0, a_1 \ldots, a_k$.
2. *The set $\mathcal{V}_D$ of finite sups of step functions $\mathbf{I}[-K, K]^n \to \mathbf{I}[-M, M]^n$,*
$$f = \bigsqcup_{1 \le j \le k} \beta_j \searrow \gamma_j \text{ where } \beta \searrow \gamma(x) = \begin{cases} \gamma & x \subseteq \beta^o \\ [-M, M]^n & \text{otherwise} \end{cases}$$
3. *For any f as above, we put $\mathcal{N}(f) = k$ and call it the* complexity of representation *of f.*

The set of partitions Q with partition points in D is denoted by $\mathcal{P}_D$; we write $\mathcal{N}(Q) = k$ if $Q = (q_0, \ldots, q_k)$ has $k + 1$ partition points.

It is easy to see that $\mathcal{S}_D$ and $\mathcal{V}_D$ are bases of the dcpos $\mathcal{S}$ and $\mathcal{V}$, respectively.

Fact 17. *If $D \subseteq \mathbb{R}$ is a dense subset, then $\mathcal{S}_D$ and $\mathcal{V}_D$ are bases of $\mathcal{S}$ and $\mathcal{V}$.*

For a particular dense subset $D \subseteq \mathbb{R}$, such as the rational or dyadic numbers, the elements of $\mathcal{S}_D$ and $\mathcal{V}_D$ are data types, the elements of which can be manipulated without loss of precision. This allows us to guarantee the convergence speed also for an implementation of our method. We now show that in the computation of $E_u(Q)$ these data types are actually preserved.

Proposition 18. *Suppose $D \subseteq \mathbb{R}$ is dense.*

1. *If D is a ring and $u \in \mathcal{V}_D$, $Q \in \mathcal{P}_D$, then $E^c_u(Q) \in \mathcal{S}_D$ and $E^c_u(Q)$ can be computed in $\mathcal{O}(\mathcal{N}(Q) \cdot \mathcal{N}(u))$ algebraic steps.*
2. *If D is a field and $u \in \mathcal{V}_D$, $Q \in \mathcal{P}_D$, then $E^l_u(Q) \in \mathcal{S}_D$ and can be computed in $\mathcal{O}(\mathcal{N}(Q) \cdot \mathcal{N}(u)^2))$ algebraic steps.*

This proposition in particular highlights the difference between the two operators E^c and E^l: computing with E^l yields a better speed of convergence to the solution (Corollary 15), at the cost of a higher complexity of the computation of the approximate solution. Furthermore, we have to work with rational (as opposed to dyadic numbers) when implementing the method using the operator E^l, as the base we need to work with is constructed using a dense subfield of the real numbers.

As we have seen, our methods for computing solutions of initial value problems hinges on the fact that we can actually produce approximations u_n to u of the form $\bigsqcup_{1 \le j \le l} \beta_j \searrow \gamma_j$. For a classical vector field $v : [-K, K]^n \to [-M, M]^n$, such approximations can be produced given a function $\hat{v}$ that computes rational approximations of v up to any desired degree of accuracy, i.e. $\hat{v} : [-K, K]^n \cap \mathbb{Q}^n \times \mathbb{Q} \to [-M, M]^n \cap \mathbb{Q}^n$ for which $\|v(x) - \hat{v}(x, \epsilon)\| \le \epsilon$. For many functions, e.g. polynomials or analytic functions, such approximating functions are both known and easy to implement.

Given $\hat{v}$ as above, we can use the Lipschitz constant L of v and the error bound to approximate an interval extension of v by finite suprema of step functions of the form $(\{b\} \oplus \delta) \searrow \{\hat{v}(b, \epsilon)\} \oplus (\epsilon + \delta L)$, where δ and ϵ vary over positive real numbers and $b \in [-K, K]^n$. In the term $\{\hat{v}(b, \epsilon)\} \oplus (\epsilon + \delta L)$, ϵ is needed to accommodate the error of $\hat{v}$ and expanding further with δL uses the Lipschitz constant of v to give a guaranteed enclosure of the values of v on the interval $\{b\} \oplus \delta$. This is developed in the full version of this paper.

6 Conclusions and Further Work

We have presented a domain theoretic method for solving initial value problems, with the domain of intervals at the heart of our approach. The main difference to other interval methods [3,9] is that we use approximations of the vector field in the process of computing solutions. As these approximations are elements of proper data types, no loss of precision is incurred when working with these approximations, allowing us to guarantee convergence also for implementations.

From the perspective of domain theory, differential equations have been studied in [4,5], using a Picard operator. This requires us to store approximative solutions in memory before being able to compute a further iterate. In comparison, the method outlined in this paper is more memory effective.

Differential equations have also been considered in the framework of exact computation, e.g. [15,10], but to our knowledge, this not has lead to practical implementations of methods for solving IVPs with guaranteed error bounds.

Finally, we remark that this is only part of a first investigation for using domain theoretic methods in the context of ODE solving. Further work is needed to be able to exploit information about the derivatives of the vector field. Also, our approach does not include any control over the step size (distance between successive partition points), but we believe that the standard techniques developed in interval analysis fit in smoothly to our framework. On the practical side, our next task is to compare implementations of our method to traditional interval based approaches, such as Lohner's AWA [12] and Nedialkov's VNODE [16].

References

1. S. Abramsky and A. Jung. Domain Theory. In S. Abramsky, D. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3. Clarendon Press, 1994.
2. M. Berz and K. Makino. Verified integration of odes and flows using differential algebraic methods on high-order taylor models. *Reliable Computing*, 4:361–369, 1998.
3. G. Corliss. Survey of interval algorithms for ordinary differential equations. *Journal of Applied Mathematics and Computation*, 31:112–120, 1989.
4. A. Edalat, M. Krznarić, and A. Lieutier. Domain-theoretic solution of differential equations (scalar fields). In *Proceedings of MFPS XIX*, volume 83 of *Elect. Notes in Theoret. Comput. Sci.*, 2004.
5. A. Edalat and D. Pattinson. A domain theoretic account of picard's theorem. In *Proc. ICALP 2004*, Lect. Notes in Comp. Sci., 2004.
6. G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. Scott. *Continuous Lattices and Domains*. Cambridge University Press, 2003.
7. A. Iserles. *Numerical Analysis of Differential Equations*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 1996.
8. K. Jackson and N. Nedialkov. Some recent advances in validated methods for ivps for odes. *Applied Numerical Mathematics archive*, 42(1):269–284, 2002.
9. K. Jackson, N. Nedialkov, and G.Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105:21–68, 1999.
10. K. Ko. On the computational complexity of ordinary differential equations. *Inform. Contr.*, 58:157–194, 1983.
11. R. Lohner. Enclosing the solution of ordinary initial and boundary value problems. In E. Kaucher, U. Kulisch, and C. Ullrich, editors, *Computer Arithmetic: Scientific Computation and Programming Languages*, Wiley-Teubner Series in Computer Science. 1987.
12. R. Lohner. *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*. PhD thesis, 1988.
13. R. Moore. *Methods and Applications of Interval Analysis*. Number 2 in SIAM studies in applied mathematics. SIAM, 1979.
14. R. E. Moore. *Interval Analysis*. Prentice-Hall, 1966.
15. N. Müller and B. Moiske. Solving initial value problems in polynomial time. In *In Proceedings of the 22th JAIIO - Panel'93*, pages 283–293, 1993.
16. N. Nedialkov and K.Jackson. The design and implementation of an object-oriented validated ode solver. Draft, available via `http://www.cas.mcmaster.ca/~nedialk/`.

Reliable Computation of Equilibrium States and Bifurcations in Nonlinear Dynamics

C. Ryan Gwaltney and Mark A. Stadtherr*

Department of Chemical and Biomolecular Engineering, 182 Fitzpatrick Hall
University of Notre Dame, Notre Dame IN 46556, USA
markst@nd.edu

Abstract. A problem of frequent interest in the analysis of nonlinear ODE models is the location of equilibrium states and bifurcations. Interval-Newton techniques are explored for identifying, with certainty, all equilibrium states and all codimension-1 and codimension-2 bifurcations of interest within specified model parameter intervals. The methodology is demonstrated using a tritrophic food chain in a chemostat (Canale's model), and a modification thereof.

1 Introduction

A problem of frequent interest in many fields of science and engineering is the study of nonlinear dynamics. Through the use of bifurcation diagrams, a large amount of information concerning the number and stability of equilibria in a nonlinear ODE model can be concisely represented. Bifurcations of equilibria can be found by solving a nonlinear algebraic system consisting of the equilibrium (steady-state) conditions along with one or more augmenting functions. Typically this equation system is solved using some continuation-based tool (e.g., AUTO). However, in general, these methods do not provide any guarantee that all bifurcations will be found, and are often initialization dependent. Thus, without some *a priori* knowledge of system behavior, one may not know with complete certainty if all bifurcation curves have been identified and explored. We consider here the use of an interval-Newton methodology as a way to ensure that *all* equilibrium states and bifurcations of interest are found.

In particular, we are interested in locating equilibrium states and bifurcations in food chain models. These models are descriptive of a wide range of behaviors in the environment, and are useful as a tool to perform ecological risk assessments. These models are often simple, but display rich mathematical behavior, with varying numbers and stability of equilibria that depend on the model parameters (e.g., [1,2]). Therefore, bifurcation analysis is quite useful in characterizing the mathematical behavior of predator/prey systems, as it allows for the concise representation of model behavior over a wide range of parameters. We will focus on one particular food chain model here, namely Canale's chemostat model. We will also develop and study a version of the model that incorporates an ecosystem contaminant.

Our interest in ecological modeling is motivated by its use as one tool in studying the impact on the environment of the industrial use of newly discovered materials. Clearly

* Author to whom all correspondence should be addressed. Fax: (574) 631-8366

it is preferable to take a proactive, rather than reactive, approach when considering the safety and environmental consequences of using new compounds. Of particular interest is the potential use of room temperature ionic liquid (IL) solvents in place of traditional solvents [3]. IL solvents have no measurable vapor pressure and thus, from a safety and environmental viewpoint, have several potential advantages relative to the traditional volatile organic compounds (VOCs) used as solvents, including elimination of hazards due to inhalation, explosion and air pollution. However, ILs are, to varying degrees, soluble in water, thus if they are used industrially on a large scale, their entry into the environment via aqueous waste streams is of concern. The effects of trace levels of ILs in the environment are today essentially unknown and thus must be studied. Ecological modeling provides a means for studying the impact of such perturbations on a localized environment by focusing not just on the impact on one species, but rather on the larger impacts on the food chain and ecosystem.

2 Problem Formulation

2.1 Canale's Chemostat Model

Canale's chemostat model is a tritrophic (prey, predator, superpredator) food chain model embedded in a chemostat. The predator and superpredator grow by consuming the prey and predator species, respectively, while the prey grows by consuming nutrients in the chemostat. The rate at which the prey, predator, and superpredator consume food is modeled by a hyperbolic functional response. There is a constant flow through the chemostat, which carries nutrients into and out of the system. The model is given by:

$$\frac{dx_0}{dt} = D(x_n - x_0) - \frac{a_1 x_0 x_1}{b_1 + x_0} \tag{2.1}$$

$$\frac{dx_1}{dt} = x_1 \left[e_1 \frac{a_1 x_0}{b_1 + x_0} - \frac{a_2 x_2}{b_2 + x_1} - d_1 - \varepsilon_1 D \right] \tag{2.2}$$

$$\frac{dx_2}{dt} = x_2 \left[e_2 \frac{a_2 x_1}{b_2 + x_1} - \frac{a_3 x_3}{b_3 + x_2} - d_2 - \varepsilon_2 D \right] \tag{2.3}$$

$$\frac{dx_3}{dt} = x_3 \left[e_3 \frac{a_3 x_2}{b_3 + x_2} - d_3 - \varepsilon_3 D \right] \tag{2.4}$$

Here x_0 is the nutrient concentration in the system and x_1, x_2, and x_3 are the biomasses of the prey, predator, and superpredator populations, respectively. The parameters a_i, b_i, d_i, and e_i are the maximum predation rate, half-saturation constant, density-dependent death rate, and predation efficiency of the prey $(i = 1)$, predator $(i = 2)$, and superpredator $(i = 3)$ species. The parameter x_n is the nutrient concentration flowing into the system, and the parameter D is the inflow rate (equal to the outflow rate). The term $\varepsilon_i D$ is the density-dependent washout rate of species i. This model has received considerable attention in the field of theoretical ecology [2,4].

2.2 Equilibrium States and Bifurcations

The equilibrium (steady-state) condition is simply $d\mathbf{x}/dt = \mathbf{0}$, which in this case is subject to the feasibility condition $\mathbf{x} \geq \mathbf{0}$. Here $\mathbf{x} = [x_0, x_1, x_2, x_3]^{\mathrm{T}}$ and $d\mathbf{x}/dt$ is given by Eqs. (2.1–2.4). Thus, once all the model parameters have been specified, there is a 4×4 system of nonlinear equations to be solved for the equilibrium states. In general, equation systems of this type may have multiple solutions, and the number of equilibrium states may be unknown *a priori*. For simple models, it may be possible to solve for many of the equilibrium states analytically, but for more complex models a computational method is needed that is capable of finding, with certainty, all the solutions of the nonlinear equation system.

A bifurcation is a change in the topological type of the phase portrait as one or more model parameters are varied [5]. Bifurcations of interest here occur at parameter values where the number and/or stability of equilibrium states change. These include three types of codimension-1 bifurcations, namely fold, transcritical and Hopf, and two types of codimension-2 bifurcations, namely double-fold (or double-zero) and fold-Hopf.

When a fold or transcritical bifurcation of equilibrium occurs, two equilibria “collide" as the bifurcation parameter is varied. This collision results in either an exchange of stability (transcritical) or mutual annihilation of two equilibria (fold). Mathematically, when an equilibrium state undergoes either a fold or transcritical bifurcation, an eigenvalue of its Jacobian is zero. Since the determinant of a matrix is equal to the product of its eigenvalues, the determinant of the Jacobian will be zero at a fold or transcritical bifurcation, thereby providing a convenient test function [5]. Thus, to locate fold or transcritical bifurcations of equilibria, the equilibrium condition can be augmented with the additional equation $\det[J(\mathbf{x}, \alpha)] = 0$ and additional variable α, the free parameter. The augmented system is then solved to find the fold and transcritical bifurcations, along with the corresponding value of α.

When a Hopf bifurcation occurs, an equilibrium state simply changes stability. Mathematically, when an equilibrium state undergoes a Hopf bifurcation, its Jacobian has a pair of complex conjugate eigenvalues whose real parts are zero. Thus, there must be a pair of eigenvalues that sums to zero. According to Stephanos's theorem [5], for an $N \times N$ matrix J with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_N$, the bialternate product $J \odot J$ has eigenvalues $\lambda_i \lambda_j$ and the bialternate product $2J \odot I$ has eigenvalues $\lambda_i + \lambda_j$. Thus, to locate a Hopf bifurcation, the equilibrium condition can be augmented with the additional equation $\det[2J(\mathbf{x}, \alpha) \odot I] = 0$. Note that while solutions to the augmented system will include all Hopf bifurcation points, there may be other solutions corresponding to the case of two eigenvalues that are real additive inverses. To identify such “false positives" it is necessary to compute the eigenvalues of the Jacobian at each solution. If the Hopf bifurcation occurs in an independent two-variable subset of state space, this is referred to as a planar Hopf bifurcation.

The two types of codimension-2 bifurcations (double-fold and fold-Hopf) can both be located by using the same augmenting functions as introduced above. When an equilibrium undergoes a double-fold bifurcation, its Jacobian has two zero eigenvalues. When an equilibrium undergoes a fold-Hopf bifurcation, its Jacobian has one eigenvalue that is zero and a pair of purely imaginary complex conjugate eigenvalues. Thus, the determinant of the Jacobian will be zero in both a double-fold and a fold-Hopf bifurcation, because in both cases there is at least one eigenvalue that is zero. Furthermore, in both cases, there is a pair of eigenvalues that will sum to zero, and so the determinant of the bialternate product $2J \odot I$ will be zero. Thus, to locate a double-fold or a fold-Hopf codimension-two bifurcation of equilibrium, the equilibrium condition can be augmented with the two additional equations $\det[J(\mathbf{x}, \alpha, \beta)] = 0$ and $\det[2J(\mathbf{x}, \alpha, \beta) \odot I] = 0$ and two additional variables (free parameters) α and β. The augmented system is then solved to find the codimension-2 bifurcations of interest, along with the corresponding values of α and β. Once found, these solutions must be screened for solutions that have a pair of (nonzero) eigenvalues that are purely real additive inverses, and the solutions must be further sorted and classified by type.

Whether one is looking for equilibrium states, or the bifurcations of equilibria discussed above, there is a system of nonlinear equations to be solved that may have multiple solutions, or no solutions, and the number of solutions may be unknown *a priori*. A computational method is needed that is capable of finding, with certainty, all the solutions of these nonlinear equation systems. An interval-Newton methodology is explored here for this purpose.

3 Computational Methodology

Consider an $n \times n$ nonlinear equation system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ with a finite number of real roots in some initial interval $\mathbf{X}^{(0)}$. The interval-Newton methodology (e.g., [6,7]) is applied to a sequence of subintervals of $\mathbf{X}^{(0)}$. For a subinterval $\mathbf{X}^{(k)}$ in the sequence, the first step is the *function range test*. An interval extension $\mathbf{F}(\mathbf{X}^{(k)})$ of the function $\mathbf{f}(\mathbf{x})$ is calculated, which provides upper and lower bounds on the range of values of $\mathbf{f}(\mathbf{x})$ in $\mathbf{X}^{(k)}$. If there is any component of the interval extension $\mathbf{F}(\mathbf{X}^{(k)})$ that does not include zero, then the interval can be discarded. Additional tools (e.g., constraint propagation) may also be applied at this point in order to reduce the size of $\mathbf{X}^{(k)}$ or eliminate it.

If it has not been eliminated, the testing of $\mathbf{X}^{(k)}$ continues with the *interval-Newton test*, which involves the solution of a linear interval equation system. There are three possible outcomes: 1. $\mathbf{X}^{(k)}$ is shown to contain no root, so it can be discarded; 2. $\mathbf{X}^{(k)}$ is shown to contain a unique root, so it need not be further tested; 3. Neither of the above, but the size of $\mathbf{X}^{(k)}$ may have been reduced. In the last case, if there has been a significant reduction is size, then the interval-Newton test can be reapplied. Otherwise, the current $\mathbf{X}^{(k)}$ is bisected, and the resulting two subintervals are added to the sequence of subintervals to be tested. If an interval containing a unique root has been identified, then this root can be tightly enclosed by continuing the interval-Newton iteration, which will converge quadratically to a desired tolerance. This approach is referred to as an interval-Newton/generalized-bisection (IN/GB) method. At termination, when the subintervals in the sequence have all been tested, either all the real roots of $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ have been tightly enclosed or it is determined rigorously that no roots exist. An important feature of this approach is that, unlike standard methods for nonlinear equation solving that require a *point* initialization, the IN/GB methodology requires only an initial *interval*.

4 Results: Canale's Model

Following Gragnani *et al.* [2], the parameters used are set to $a_1 = 1.25$, $b_1 = 8$, $e_1 = 0.4$, $d_1 = 0.01$, $\varepsilon_1 = 1$, $a_2 = 0.33$, $b_2 = 9$, $e_2 = 0.6$, $d_2 = 0.001$, $\varepsilon_2 = 0.8$, $a_3 = 0.021$,

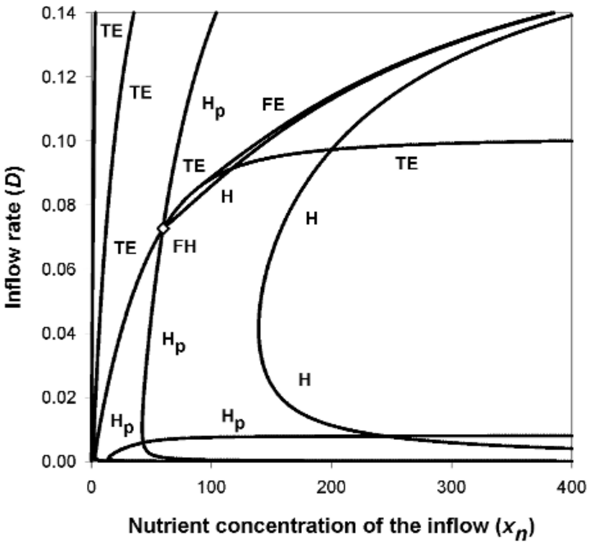


Fig. 1. Bifurcation of equilibrium diagram of nutrient inflow concentration (x_n) versus inflow rate (D) for Canale's chemostat model. TE: Transcritical of equilibrium; FE: Fold of equilibrium; H: Hopf; $\mathrm{H_p}$: Planar Hopf; FH: Fold-Hopf codimension-2

$b_3 = 15.19$, $e_3 = 0.9$, $d_3 = 0.0001$, $\varepsilon_3 = 0.1$. A bifurcation diagram with the inflow rate, D, and the concentration of the nutrient in the inflow, x_n, as the free parameters was then computed using the IN/GB methodology and compared to the D vs. x_n bifurcation diagram determined by Gragnani *et al.* [2] using continuation techniques. The bifurcation diagram for Canale's model computed using IN/GB is given in Fig. 1. The codimension-1 bifurcation curves were computed by solving the appropriate equation systems (see Section 2.2), first fixing x_n at many (400) closely spaced values over the interval [0,400] and determining the value(s) of D at which bifurcations occurs, and then fixing D at many (700) closely spaced values over the interval [0,0.14] and determining the value(s) of x_n at which bifurcations occurs. The average CPU time to solve a system for transcritical and fold bifurcations was about 15 seconds (1.7 GHz Xeon processor running Linux) and for Hopf bifurcations about 100 seconds (the many nonlinear systems that must be solved are independent and can be solved in parallel). Some planar Hopf ($\mathrm{H_p}$) bifurcation curves are shown (both in Fig. 1 and in Gragnani *et al.*'s diagram) for which a stability change occurs only in a two-variable subspace, with the stability of the overall system remaining unchanged (unstable) due to the sign (positive) of the third and/or fourth eigenvalue.

Fig. 1 captures all bifurcations of equilibria shown in the D vs. x_n bifurcation diagram presented by Gragnani *et al.* [2] However, we have also located other bifurcation curves not shown by Gragnani *et al.* First, we compute a transcritical bifurcation curve very near the D axis (the leftmost TE in Fig. 1) that is not given by Gragnani *et al.* At this bifurcation, a stable nutrient-only equilbrium state collides with an infeasible nutrient-prey equilibrium state; the nutrient-prey state becomes feasible and exchanges stability

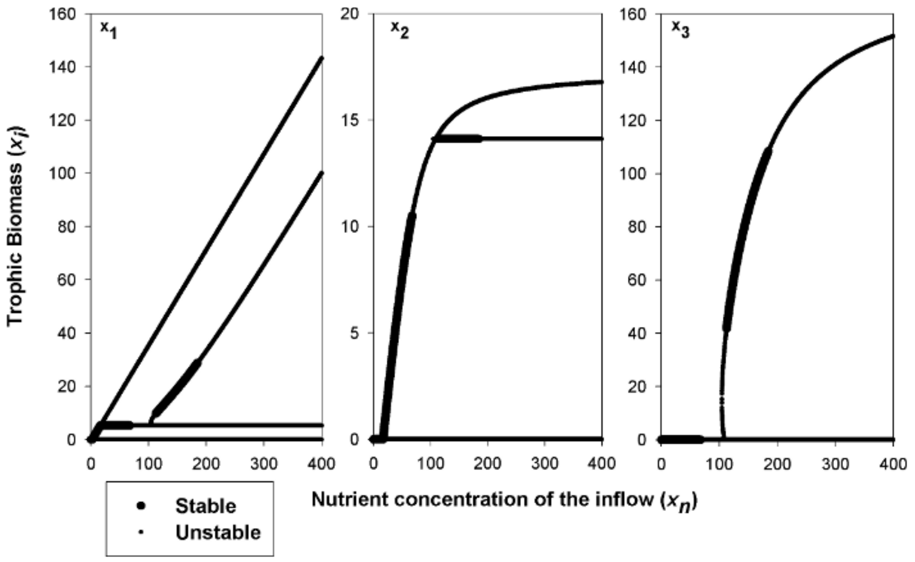


Fig. 2. Solution branch diagram illustrating the change in equilibrium states (species biomass) with change in the nutrient concentration of the inflow (x_n) for Canale's chemostat model. From left to right: prey, predator, and superpredator biomasses. $D = 0.09$ for all three plots

with the nutrient-only state. Second, we compute a planar Hopf bifurcation curve near the x_n axis (lowest H_p in Fig. 1) that is not shown by Gragnani *et al* (we have also computed other planar Hopf bifurcations curves very near the x_n axis, but these are not visible in Fig. 1 due to the scale used). However, for all of these H_p bifurcations, the stability change occurs only in a two-variable subspace, with the stability of the overall system remaining unchanged (unstable).

Another useful type of diagram in nonlinear dynamics is the solution branch diagram, which shows how the equilibrium states change as one model parameter is varied. For example, Fig. 2 shows how the equilibrium states change as the inflow nutrient concentration, x_n, is varied from 0 to 400, while the inflow rate, D, is held constant at a value of 0.09. This was computed by using IN/GB to solve the equilibrium conditions for many (4000) closely spaced values of x_n. The average CPU time to solve for the equilibrium states for one x_n value was about 0.06 seconds. In Fig. 2, thin lines represent unstable equilibria while thick lines represent stable equilibria. This figure tracks the behavior of equilibrium states as x_n is increased from 0 to 400 along the horizontal line $D = 0.09$ in Fig. 1. Moving to the right along this line, seven bifurcations are encountered, namely (and in order) TE, TE, H_p, FE, TE, H, H. The first TE is not clearly visible in Fig. 2 due to the scale used. The sixth and seventh bifurcations, both Hopf, are of particular interest here. The sixth bifurcation ($x_n \approx 112.5$) results in the first stable coexistence equilibrium state (all three species present). But at the seventh bifurcation ($x_n \approx 184.5$), this state becomes unstable. This illustrates the "paradox of enrichment." Enriching the food chain (increasing nutrient inflow) in order to increase a stable population of the top predator may be successful, but only to a point. Beyond that point (Hopf bifurcation) the system becomes unstable and the populations may experience “boom and bust" cycles.

5 Canale’s Model with Contaminant

As previously stated, our interest in ecological modeling is motivated by its use as a tool for assessing the risk of the industrial use of newly discovered materials, which may enter the environment as contaminants. A straightforward way of linking the effects of contamination to a food chain model is to directly consider the impact of a contaminant on the model parameters. For instance, one way of modeling the effect of a contaminant on a food chain would be to link the death rate parameters to experimentally measured toxicological parameters such as LC_{50}. The LC_{50} value is the concentration at which 50 percent of the organisms in a test sample die. Thus, an expression for the density dependent death rate d_i could be given by:

$$d_i = d_i^0 + \frac{C}{2C_i^{\mathrm{LC}_{50}}} \tag{5.5}$$

where d_i^0 is the baseline death rate, $C_i^{\mathrm{LC}_{50}}$ is the LC_{50} value for species i, and C is the concentration of the contaminant in the system. The effect of a contaminant on other model parameters could be described in a similar manner.

Canale’s chemostat model was modified to include the linearly increasing death rate function given by Eq. (5.5). The base-line death rates d_1^0, d_2^0, and d_3^0 were set to the d_1, d_2, and d_3 values given above [2]. The values for the LC_{50} were chosen to be $C_1^{\mathrm{LC}_{50}} = 1000$, $C_2^{\mathrm{LC}_{50}} = 10$, and $C_3^{\mathrm{LC}_{50}} = 100$. These values were chosen to reflect orders of magnitude difference in the sensitivity of each species. In this case, the predator species is particularly sensitive to the contaminant, while the superpredator is moderately sensitive and the prey species is hardly sensitive at all. A bifurcation diagram for this new model was generated using the concentration of the contaminant, C, and the nutrient inflow concentration, x_n, as the bifurcation parameters, while holding D constant at 0.07. This diagram appears in Fig. 3.

Initially one notices that the characteristics of this bifurcation diagram (Fig. 3) are quite similar to those in the top half of Fig. 1. In fact, though the shape of some of the bifurcation curves (e.g. the second TE curve from the left) is decidedly different, all of the curves appear in the same order. Thus, it appears that, at least qualitatively, increasing the contaminant concentration in the system has an effect similar to increasing the inflow rate D. Intuitively, this makes sense because increasing the inflow rate increases the washout rate, while increasing the death rate of each species effectively produces the same effect, namely removal of the organisms from the system.

Fig. 4 is a solution branch diagram that illustrates the effect of increasing the contaminant concentration in the system. Fig. 4 was generated at $D = 0.07$ and $x_n = 200$. From Fig. 4 the effects of increasing the contaminant concentration on species biomass are apparent. As the contaminant concentration increases, the system transitions to a stable, coexisting steady-state. The contaminant initially increases the death-rates of the species, causing population cycles to dampen and collapse in a Hopf bifurcation. This

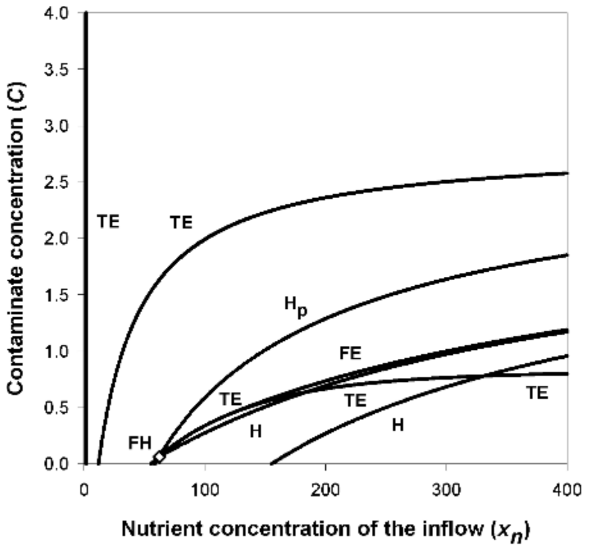


Fig. 3. Bifurcation of equilibrium diagram of nutrient inflow concentration (x_n) versus contaminant concentration (C) for Canale's chemostat model with modified death rates given by Eq. 5.5. $D = 0.07$ TE: Transcritical of equilibrium; FE: Fold of equilibrium; H: Hopf; $\mathrm{H_p}$: Planar Hopf; FH: Fold-Hopf codimension-2

stable, coexisting steady-state does become unstable prior to the decimation of the superpredator population. However, while this coexisting steady-state is stable, increasing the contaminant concentration on the system has an unexpected effect. The superpredator and prey populations both decrease while the predator population increases, despite the fact that the predator species is the most sensitive to the contaminant. This behavior is obviously counterintuitive, but is indicative of the complex interactions that often occur in food chain models. As the concentration of the contaminant increases further, a series of three bifurcations occur, beginning with a transcritical bifurcation, then a Hopf bifurcation which destabilizes the coexisting steady-state, and finally a fold bifurcation which results in the mutual annihilation of the two unstable coexisting steady-states. Past these bifurcations, another Hopf bifurcation occurs that results in a change of stability allowing a stable steady-state with only a prey and predator population to form. As the concentration of contaminant increases, the prey population increases while the predator population declines to zero. The prey population then begins to decline very slowly with increasing C.

Though not presented here, we have also used the IN/GB approach to compute other bifurcation and solution branch diagrams for Canale's chemostat model, both with and without the death rate modification, as well as for a tritrophic Rosenzweig-MacArthur model [8].

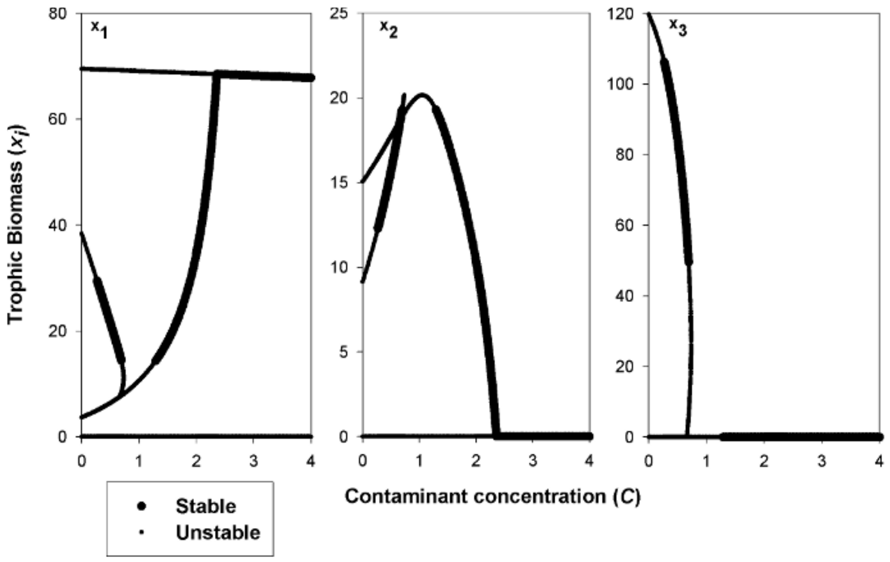


Fig. 4. Solution branch diagram illustrating the change in equilibrium states (species biomass) with change in the contaminant concentration (C) for Canale's chemostat model with modified death rates given by Eq. 5.5. From left to right: prey, predator, and superpredator biomasses. $D = 0.07$ and $x_n = 200$ for all three plots

6 Concluding Remarks

Using an interval-Newton approach, one can compute, with certainty, all equilibrium states and bifurcations of equilibria (fold, transcritical, Hopf, double-fold and fold-Hopf) in a nonlinear dynamic model. Using this methodology one can easily, without any need for initialization or *a priori* insight into system behavior, generate complete solution branch and bifurcation diagrams. While CPU intensive, since the diagrams can be generated automatically, without user intervention to deal with initialization issues, the actual elapsed time to create a new bifurcation diagram may actually be less than when initialization-dependent methods are used.

Acknowledgments

This work was supported in part by the Department of Education Graduate Assistance in Areas of National Needs (GAANN) Program under Grant #P200A010448, and by the State of Indiana 21st Century Research and Technology Fund under Grant #909010455.

References

1. Moghadas, S. M., Gumel, A. B.: Dynamical and numerical analysis of a generalized food-chain model. Appl. Math. Comput. **142** (2003) 35–49

2. Gragnani, A., De Feo, O., Rinaldi, S.: Food chains in the chemostat: Relationships between mean yield and complex dynamics. Bull. Math. Biol. **60** (1998) 703–719
3. Brennecke, J. F., Maginn, E. J.: Ionic liquids: Innovative fluids for chemical processing. AIChE J. **47** (2001) 2384–2389
4. Kooi, B. W.: Numerical bifurcation analysis of ecosystems in a spatially homogeneous environment. Acta Biotheoretica **51** (2003) 189–222
5. Kuznetsov, Y. A.: Elements of Applied Bifurcation Theory. Springer-Verlag, New York (1998)
6. Kearfott, R. B.: Rigorous Global Search: Continuous Problems. Kluwer, Dordrecht, The Netherlands (1996)
7. Gau, C.-Y., Stadtherr, M. A.: New interval methodologies for reliable chemical process modeling. Comput. Chem. Eng. **26** (2002) 827–840
8. Gwaltney, C. R., Styczynski, M. P., Stadtherr, M. A.: Reliable computation of equilibrium states and bifurcations in food chain models. Comput. Chem. Eng. **28** (2004) 1981-1996

A Verification Method for Solutions of Linear Programming Problems

Ismail I. Idriss

Institute for Applied Research
University of Applied Sciences / FH Konstanz
D-78405 Konstanz, Germany
idriss@fh-konstanz.de

Abstract. This paper is concerned with the verification of a solution of a linear programming problem obtained by an interior-point method. The presented method relies on a reformulation of the linear programming problem as an equivalent system of nonlinear equations and uses mean value interval extension of functions and a computational fixed point theorem. The designed algorithm proves or disproves the existence of a solution on a computer and, if it exists, encloses this solution in narrow bounds.

Keywords: interior-point algorithms, linear programming, interval arithmetic, verified computation.

1 Introduction

Linear programming problems have a wide range of practical applications that arise in such diverse areas as economics, computer science, operations research, medicine, finance, mathematics, as well as many branches of engineering.

There are a number of general algorithms for the solution of the linear programming problem. It can be solved very efficiently by recently developed interior-point methods. An interior-point algorithm is an iterative technique that approaches the optimal solution by generating a sequence of points in the interior of the feasible region.

The modern era of interior-point methods dates to 1984, when Karmarkar [7] proposed his new polynomial-time algorithm for linear programming. After the publication of Karmarkar's algorithm, various interior point methods were developed which are based on it. However, until now these algorithmic developments have always aimed at increasing the runtime efficiency and limiting the computational effort, not at improving the accuracy and reliability of the computational results.

Infeasible interior-point algorithms are known as the most efficient computational methods for solving linear programming problems. But these algorithms sometimes compute an approximate solution with duality gap less than a given tolerance even when the problem may not have a solution [14,17].

One approach for proving or disproving the existence and possibly also uniqueness of a solution on computers is the use of interval arithmetic, e.g., [1,8,12]. This enables data uncertainties to be considered. Moreover, by performing outward rounding all rounding errors appearing during the computation can be taken into account. As a consequence, one works with intervals of floating point numbers instead of with the numbers themselves. The computation of intervals is performed according to the rules of interval arithmetic. An algorithm implemented with interval arithemtic delivers (often tight) upper and lower bounds on the exact solution which are guaranteed also in the presence of rounding errors.

The use of interval arithmetic enables a rigorous verification of the existence and uniqueness of a solution and the computation of a guaranteed enclosure [8]. This verification is an application of Brouwer's fixed-point theorem which requires that a continuous function maps an interval vector onto itself. Such conditions can easily be tested on computers using fundamental properties of interval arithmetic. Enclosures with narrow bounds usually cannot be obtained by naively replacing real numbers by intervals and every arithmetic operation by the corresponding interval operation. One approach to improve enclosures of a differentiable function evaluated with interval arguments is to use the mean value theorem for differentiable functions in connection with an interval extension of the derivative.

The paper is organized as follows. In Section 2 we consider an infeasible interior-point algorithm for solving linear programming problems. In Section 3 we propose an algorithm that tests for the existence of a solution. Numerical results are presented in Section 4. We conclude with an empirical evaluation of our algorithm and some suggestions for further research.

In this paper, vectors are usually denoted by lower case letters, and matrices by upper case letters. Intervals and interval vectors are usually denoted by boldface lower case letters. It should be clear from the context, whether the letter denotes an interval or an interval vector. We denote the set of real numbers by $\mathbb{R}$ and the set of the compact nonempty real intervals by $\mathbb{IR}$.

2 Infeasible Interior-Point Algorithm

We consider the linear programming problem in standard form, which is generally referred to as the primal problem

$$\begin{aligned} \text{Minimize} \quad & c^\top x \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0 \end{aligned} \tag{P}$$

and its dual problem after adding the slack variable z to convert it to equality form

$$\begin{aligned} \text{Maximize} \quad & b^\top y \\ \text{subject to} \quad & A^\top y + z = c, \\ & z \geq 0 \end{aligned} \tag{D}$$

where $c, x, z \in \mathbb{R}^n$, $b, y \in \mathbb{R}^m$ are vectors and $A \in \mathbb{R}^{m \times n}$ is a matrix. Every feasible solution for one of these two problems gives considerable information about the feasibility and optimality of the other. Moreover, it can easily be shown that for a primal feasible x and dual feasible y and z,

$$c^\top x - b^\top y = x^\top z,$$

holds true. The left hand side is called the duality gap and provides an excellent and easily obtainable measure of closeness of the computed solution to the optimal one.

An infeasible interior-point algorithm solves a primal-dual pair starting from an arbitrary infeasible point that satisfies the positivity constraints, but does not necessarily satisfy the equality constraints. Then the algorithm usually generates a sequence of iterates of feasible or infeasible solutions that approach feasibility and optimality. In the limit the iterates will converge to an approximate optimal solution with duality gap less than a given tolerance. The algorithm can be derived by applying the logarithmic barrier method to the primal problem (P), or alternatively, to the dual problem (D). Here we consider the dual problem (D) which is transformed to

$$\begin{array}{ll} \text{Maximize} & b^\top y - \mu \sum_{j=1}^{n} \ln z_j \\ \text{subject to} & A^\top y + z = c, \end{array} \qquad (\mathrm{D}_\mu)$$

where $\mu > 0$ is a barrier parameter. For more details on the relation between (D) and (D$_\mu$), see [2], where the logarithmic barrier method is introduced and [3], where it is developed. The Lagrangian for (D$_\mu$) is

$$L(x, y, z, \mu) = b^\top y - \mu \sum_{j=1}^{n} \ln z_j - x^\top (A^\top y + z - c)$$

and the first order conditions for (D$_\mu$) are

$$\begin{aligned} \nabla_x L &= c - z - A^\top y = 0, \\ \nabla_y L &= b - Ax = 0, \\ \nabla_z L &= \mu Z^{-1} e - X = 0, \end{aligned} \qquad (2.1)$$

where X and Z are $n \times n$ diagonal matrices with elements x and z, respectively, and e is the n-vector of all ones. Newton's method can be applied to (1) in order to compute a search direction and then to determine a better estimate of the solution of the primal-dual pair.

Furthermore, Mehrotra [10] proposed a predictor-corrector method as an efficient strategy to compute the search direction. In particular, the method can be derived directly from the optimality conditions (2.1). Substituting (x, y, z) by $(x + \Delta x, y + \Delta y, z + \Delta z)$ leads to the implicit equations

$$\begin{aligned} A^\top \Delta y + \Delta z &= c - z - A^\top y, \\ A \Delta x &= b - Ax, \\ X \Delta z + Z \Delta x &= \mu e - XZe - \Delta X \Delta Z e, \end{aligned} \qquad (2.2)$$

where ΔX and ΔZ are $n \times n$ diagonal matrices with elements equal to Δx and Δz, respectively.

Mehrotra proposed to first solve the affine system

$$\begin{aligned} A^\top \varDelta\hat{y} + \varDelta\hat{z} &= c - z - A^\top y, \\ A\,\varDelta\hat{x} &= b - Ax, \\ X\varDelta\hat{z} + Z\varDelta\hat{x} &= -XZe, \end{aligned} \tag{2.3}$$

and then substitute the vectors $\varDelta\hat{x}$ and $\varDelta\hat{z}$ computed from (2.3) in the term $\varDelta X \varDelta Z e$ on the right-hand side of (2.2). Moreover, he suggested testing the reduction in the complementarity $(x + \alpha_P \varDelta\hat{x})^\top (z + \alpha_D \varDelta\hat{z})$, where α_P and α_D are the largest step lengths in the primal and dual variables which ensure $x > 0$ and $z > 0$. If we let

$$\hat{\mu} = (x + \alpha_P \varDelta\hat{x})^\top (z + \alpha_D \varDelta\hat{z}),$$

then a good estimate for the barrier parameter μ is

$$\mu = \left(\frac{\hat{\mu}}{x^\top}\right)^2 \left(\frac{\hat{\mu}}{n}\right).$$

This generates μ to be small when the affine direction produces a large decrease in complementarity and large otherwise.

Finally, the most current interior-point codes are based on the predictor–corrector technique [10]. Implementations of the algorithm also incorporate other heuristics that are essential for robust behaviour on many practical problems, including the determination of a starting point, the choice of step length, updating the barrier parameter μ, and the computing of the predictor–corrector direction. More details and further information on implementation techniques and the most relevant issues of infeasible interior–point methods are given in [17].

3 A Verification Method

In this section, we develop a practical verification method via reformulation of the linear programming problem as an equivalent system of nonlinear equations.

We define the function $F : \mathbb{R}^{2n+m} \to \mathbb{R}^{2n+m}$ by

$$F(x, y, z) = \begin{pmatrix} Ax - b \\ A^\top y + z - c \\ xZe - \mu e \end{pmatrix}.$$

Then we can rewrite conditions (2.1) in the form $F(x, y, z) = 0$. The Jacobian of F is given by

$$J_F(x, y, z) = \begin{pmatrix} A & 0 & 0 \\ 0 & A^\top & I \\ Z & 0 & x \end{pmatrix},$$

which is nonsingular for all $(x, z) > 0$.

By using the notation

$$u = (x, y, z),$$
$$\mathbf{u} = \{(u_1, u_2, \ldots, u_{2n+m})^\top \in \mathbb{R}^{2n+m} \mid \underline{u}_i \le u_i \le \overline{u}_i,\ 1 \le i \le 2n+m\},$$

the problem is then to verify that there exists a $u^* \in \mathbf{u}$ such that $F(u^*) = 0$. Interval arithmetic in combination with the mean value extension and Brouwer's fixed point theorem can be used to verify the existence and uniqueness of a solution to $F(u) = 0$ (see [8]).

The mean value extension can be constructed using both the mean value theorem and the interval extension of the derivative.

Theorem 1 (Mean value extension). *Let $f : D \subseteq \mathbb{R}^n \to \mathbb{R}$ be a continuously differentiable function and let the gradient ∇f have an interval extension for all $\mathbf{u}$ contained in the domain of f. Then*

$$f(u) \in f(\tilde{u}) + \nabla f(\mathbf{u})\,(\mathbf{u} - \tilde{u}) \quad \textit{for all} \quad u, \tilde{u} \in \mathbf{u}.$$

Proof. For a proof see, for example, [1].

A computationally verifiable sufficient condition is given by Krawczyk [9] for the existence of a solution to a system of nonlinear equations using an interval version of Newton's method.

Definition 1. *Let $F : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ be a continuously differentiable function and let $\mathbf{u}$ be an interval vector in $\mathbb{IR}^n$ with $\mathbf{u} \subseteq D$. Then shape Krawczyk's operator is defined as:*

$$K(\mathbf{u}, \tilde{u}) = \tilde{u} - R \cdot F(\tilde{u}) + (I - R \cdot J_F(\mathbf{u}))(\mathbf{u} - \tilde{u}),$$

where $\tilde{u}$ is a vector in $\mathbf{u}$, $J_F(\mathbf{u})$ is an interval evaluation of the Jacobian of F, R is a real nonsingular matrix approximating $(J_F(\tilde{u}))^{-1}$ and I is the $n \times n$ identity matrix.

A verification method can be derived by iteratively applying Krawczyk's operator until the conditions of the following theorem are satisfied:

Theorem 2 (Existence and uniqueness). *If $K(\mathbf{u}, \tilde{u}) \subseteq \mathbf{u}$, then F has a zero u^* in $K(\mathbf{u}, \tilde{u})$ and therefore also in $\mathbf{u}$.*

Proof. For a proof see, for example, [9] or [11].

Let a continuously differentiable function F and an interval vector $\mathbf{u}$ be given. The midpoint of $\mathbf{u}$ is defined as $\text{mid}(\mathbf{u}) = (\underline{\mathbf{u}} + \overline{\mathbf{u}})/2$. A verification algorithm for testing the existence or nonexistence of a solution of $F(u) = 0$ in $\mathbf{u}$ is based on a sequence of iterates of the form:

$$\begin{aligned}
\mathbf{u}^0 &= \mathbf{u}, \\
\tilde{u}^k &= \text{mid}(\mathbf{u}^k), \\
\mathbf{u}^{k+1} &= K(\mathbf{u}^k, \tilde{u}^k) \cap \mathbf{u}^k, \quad k = 0, 1, 2, \ldots
\end{aligned}$$

Our numerical experiences show that during the first iteration it can often be decided whether there is a zero of F in $\mathbf{u}$, if $\mathbf{u}$ is sufficiently small. Sharper interval bounds may be obtained by continuing the iteration. Moreover, an important and powerful result is that we do not need to solve a system of linear equations to find an enclosure of zeroes of a system of nonlinear equations. More details and further information on this algorithm can be found in [9], where the method was first introduced, and in [11], where the power of the method as a computational fixed point theorem was first analyzed.

Furthermore, the following properties of this algorithm are remarkable, see also [8,12], where its theory and practice have been expounded. Here it is assumed that $\mathrm{mid}\big(J_F(\mathbf{u}^k)\big)$ is nonsingular.

1. Every zero of F in $\mathbf{u}$ can be computed and correctly bounded.
2. If there is no zero of F in $\mathbf{u}$, the algorithm will automatically prove this fact in a finite number of iterations.
3. The proof of existence or nonexistence of a zero of F in a given interval vector $\mathbf{u}$ can be delegated to the computer. If $K(\mathbf{u},\tilde{u}) \overset{\circ}{\subset} \mathbf{u}$, then there exists a unique zero of F in $\mathbf{u}$. If $K(\mathbf{u},\tilde{u}) \cap \mathbf{u} = \emptyset$, then there is no zero of F in $\mathbf{u}$. Here, the relation $\overset{\circ}{\subset}$ denotes inner inclusion, i.e., $K(\mathbf{u},\tilde{u}) \overset{\circ}{\subset} \mathbf{u}$ states that the interval vector $K(\mathbf{u},\tilde{u})$ is contained in the interior of $\mathbf{u}$.
4. In practice, the algorithm converges quadratically when $\tilde{u}$ is taken to be the midpoint vector of $\mathbf{u}$, R is an appropriate approximation to the midpoint inverse, the interval extensions are sufficiently sharp, and the widths of the components of $\mathbf{u}$ are sufficiently small.

Now, the general algorithm for verification of approximate solutions of linear programming problems can be stated as follows:

ALGORITHM (General verification algorithm)

Step 0. Let a primal problem (P) and its dual (D) be given.

Step 1. Compute an approximate solution $\tilde{u} = (\tilde{x}, \tilde{y}, \tilde{z})$ for the primal problem (P) and its dual (D) by running an infeasible interior-point algorithm.

Step 2. Construct a small box $\mathbf{u} = (\mathbf{x}, \mathbf{y}, \mathbf{z})$ around the approximate solution by using epsilon-inflation [13]

$$\mathbf{u} = \tilde{u} + \epsilon\,[-e, e], \tag{3.4}$$

where ϵ is a given small positive machine number and e is the $(2n+m)$-vector of all ones.

Step 3. Perform a verification step:

Step 3.1. Prove the existence or nonexistence of a zero by applying Krawczyk's method to the nonlinear system $F(u) = 0$.

Step 3.2. Enclose the optimal solution value by the interval

$$c^\top \mathbf{x} \cap b^\top \mathbf{y}. \tag{3.5}$$

We conclude with some practical remarks on the algorithm. If the optimality conditions (2.1) hold and the existence of an optimal feasible solution in the constructed box is proved, then bounds on the exact optimal objective value are provided by (3.5). However, the approximate solution $\tilde{u}$ should be near an exact solution. Thus, if the tolerance of the stopping criterion in the approximate algorithm is set smaller than ϵ (3.4), then the verification step will succeed. Finally, if the infeasible interior-point algorithm fails to compute a solution to the primal-dual pair, then the verification algorithm requires an initial interval vector approximation to be prescribed by the user. It then manages to contract this initial interval vector.

4 Numerical Results

In this section, we evaluate the practical efficiency of the presented algorithm. To demonstrate the effects of our verification algorithm, we have compared it against the software package LOQO [15] which is a linear programming solver based on an infeasible primal-dual interior-point algorithm. The computations are performed on a personal computer using FORTRAN–XSC [16]. In all our examples the quantity ϵ in (3.4) is set to 0.25. For more details on the efficient use of epsilon–inflation in verification methods, refer to [13].

Several software tools have been developed over the last decades to improve the accuracy of computer arithmetic. They include the library FORTRAN–XSC for interval arithmetic which provides reliable arithmetic by producing two values for each result. These two values correspond to the endpoints of an interval such that the exact result is guaranteed to be contained within it. Moreover, FORTRAN–XSC provides special notations and data types for interval arithmetic and an exact dot product which is particularly suited to the design of algorithms delivering automatically verified results with high accuracy. For more details, see [16], where the concepts and the important features are described.

Example 1. Consider the following linear programming problem

$$\begin{array}{lll}\text{Minimize } -50x_1 - 9x_2 + 10^{-40}x_3 & & \\ \text{subject to} \qquad\qquad x_1 & & \le 50 \\ \qquad\qquad\qquad\qquad x_2 & & \le 200 \\ \qquad\qquad 100x_1 + 18x_2 & & \le 5000 \\ \qquad\qquad\qquad x_1, x_2 & & \ge 0.\end{array}$$

This example is taken from [4], page 221, where the exact optimal value enclosure,

$$[-2.500000000000000 \cdot 10^3, -2.500000000000000 \cdot 10^3],$$

is computed by using the verified simplex method from [6]. The results achieved by running the infeasible interior-point algorithm [15] for this problem show that the primal problem and its dual are infeasible. Our verification algorithm proves that this problem has a solution. The optimal value is enclosed in the interval

$$[-2.5000000000000019 \cdot 10^3, -2.4999999999999981 \cdot 10^3].$$

A comparison between the results obtained for this test problem indicates that the approximate result from LOQO is quantitatively, but also qualitatively incorrect.

Example 2. Consider the following linear programming problem given in the form (P) with

$$A = \begin{pmatrix} -2 & -1 & -2 \\ -1 & -2 & -2 \\ -1 & -3 & -1 \end{pmatrix}, \quad b = \begin{pmatrix} -3 \\ -1 \\ -3 \end{pmatrix}, \quad \text{and} \quad c = \begin{pmatrix} 2 \\ 5 \\ 6 \end{pmatrix}.$$

This linear programming problem has no feasible solution, and this fact is proved by our verification algorithm. LOQO computes after 9 iterations an approximate optimal solution of this problem

$$= \begin{pmatrix} 2.6 \\ 0.6 \\ -1.4 \end{pmatrix}$$

with the optimal objective values,

$$\begin{aligned} \text{primal objective} &= -0.2000000028 \\ \text{dual objective} &= -0.1999999998\,. \end{aligned}$$

The infeasibility of this solution is not recognized by this algorithm. By constructing a small box $\mathbf{u}$ around the obtained approximate solution using epsilon-inflation and performing a verification step, we obtain that

$$K(\mathbf{u}, \tilde{u}) \cap \mathbf{u} = \emptyset.$$

Hence, the linear programming problem has no solution in $\mathbf{u}$ in spite of the fact that an approximate optimal solution with optimal objective values has been computed by LOQO. Therefore, our verification method provides a safety valve for the infeasible interior-point algorithm.

Example 3. Consider the following linear programming problem

$$\begin{aligned} \text{Minimize} \quad & 2x_1 + x_2 \\ \text{subject to} \quad & -x_1 - x_2 = 10^{-6}, \\ & x_1, x_2 \geq 0. \end{aligned}$$

When our algorithm attempts to verify the existence of a solution for this problem, it affirms that the problem is infeasible. In contrast, LOQO terminated after 254 iterations with the result that the primal problem and its dual are infeasible. This result from LOQO typically specifies that a maximum number of iterations is performed and no optimal solution has been computed.

5 Conclusions

We conclude with some observations and propose directions for future research.

The main goal was to develop an effective verification method for solutions of linear programming problems which computes enclosures for the solutions also in the presence of roundoff errors.

In combination with the fundamental properties of interval arithmetic, other tools such as Brouwer's fixed point theorem are used to automatically compute rigorous bounds on exact solutions.

The infeasible interior-point method and the stopping criteria have been realized in floating point arithmetic. The verification algorithm has been implemented using FORTRAN–XSC. The potential of this approach has been documented by comparing the results for three linear programming problems. Our results for many other larger problems up to a hundred variables indicate that the presented algorithm is robust [5]. A reliable result was obtained for all test problems.

We outline some directions for further research. It is essential for the infeasible interior-point algorithm to have a good starting point. The choice of starting point has a strong influence on the number of iterations, and a bad choice may lead to divergence or cycling so that a solution can never be reached. The best choice for a default starting point is still very much an open question. This problem concerns not only the verification algorithm but is more generally a problem of any infeasible interior-point algorithm. Further research for algorithmic improvements should be aimed at accelerating the verification process. In particular, the computation of an approximation to the midpoint inverse usually requires a lot of runtime. Finally, although interior-point methods have been extended to more general classes of problems, verification algorithms for these have not yet appeared.

Acknowledgment

The author would like to thank Jürgen Garloff and Andrew P. Smith for the careful reading of the manuscript and numerous suggestions for improvements in the presentation. Support from Damascus University, the Ministry of High Education of Syria, and the Ministry of Education and Research of Germany under contract no. 1705803 is gratefully acknowledged.

References

1. G. E. Alefeld and J. Herzberger, Introduction to Interval Computations, Academic Press, New York and London, 1983.
2. R. K. Frisch, The logarithmic potential method of convex programming, Memorandum, University Institute of Economics, Oslo, 1955.
3. A. V. Fiacco and G. P. McCormick, Nonlinear Programming: Sequential Unconstraint Minimization Techniques. SIAM Classics in Applied Mathematics, vol. 4 (1990), Philadelphia.
4. R. Hammer, M. Hocks, U. Kulisch and D. Ratz, Numerical Toolbox for Verified Computing I, Springer-Verlag, Berlin, 1993.
5. I. I. Idriss, Verified Linear Programming in Control System Analysis and Design, Dresden University of Technology, Dissertation, 2001.

6. C. Jansson, Zur Linearen Optimierung mit unscharfen Daten. Dissertation, Universität Kaiserslautern, 1985.
7. N. K. Karmarkar, A New Polynomial-Time Algorithm for Linear Programming, *Combinatorica 4* (1984) 373-395.
8. R. B. Kearfott, Rigorous Global Search: Continuous Problems, Kluwer Academic Publishers, Dordrecht, Netherlands, 1996.
9. R. Krawczyk, Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken, *Computing 4* (1969) 187–201.
10. S. Mehrotra, On the Implementation of a Primal-Dual Interior Point Method, *SIAM Journal on Optimization 2:4* (1992) 575–601.
11. R. E. Moore, A Test for Existence of Solutions to Nonlinear Systems, *SIAM Journal on Numerical Analysis 14* (1977) 611-615.
12. A. Neumaier, Interval Methods for Systems of Equations, Cambridge University Press, London, 1990.
13. S. M. Rump, Verification Methods for Dense and Sparse Systems of Equations, Topics in Validated Computations — Studies in Computational Mathematics, J. Herzberger , editor, Elsevier Amsterdam, (1994) 63–136.
14. M. J. Todd, Detecting Infeasibility in Infeasible-Interior-Point Methods for Optimization, in: *Foundations of Computational Mathematics*, F. Cucker, R. DeVore, P. Olver and E. Suli, (editors) Cambridge University Press (2004) 157–192.
15. R. J. Vanderbei, Loqo User's Manual, Version 4.05, Technical Report NO. ORFE-99-307, Department of Operation Research and Financial Engineering, Princeton University, 2000.
16. W. V. Walter, FORTRAN-XSC: A Portable Fortran 90 Module Library for Accurate and Reliable Scientific Computing, in Validation Numerics — Theory and Application, R. Albrecht, G. Alefeld and H. J. Stetter (eds.), Computing Supplementum 9 (1993) 265–285.
17. S. J. Wright, Primal-Dual Interior-Point Methods, SIAM Publication, Philadelphia, 1997.

Compressing 3D Measurement Data Under Interval Uncertainty

Olga Kosheleva[1], Sergio Cabrera[1], Brian Usevitch[1], and Edward Vidal Jr.[2]

[1] Department of Electrical and Computer Engineering
University of Texas, El Paso, TX 79968, USA
olgak@utep.edu
[2] Army Research Laboratory, While Sands Missile Range, NM 88002-5501, USA
evidal@arl.army.mil

Abstract. The existing image and data compression techniques try to minimize the mean square deviation between the original data $f(x, y, z)$ and the compressed-decompressed data $\widetilde{f}(x, y, z)$. In many practical situations, reconstruction that only guaranteed mean square error over the data set is unacceptable.
For example, if we use the meteorological data to plan a best trajectory for a plane, then what we really want to know are the meteorological parameters such as wind, temperature, and pressure along the trajectory. If along this line, the values are not reconstructed accurately enough, the plane may crash – and the fact that on average, we get a good reconstruction, does not help.
In general, what we need is a compression that guarantees that for each (x, y), the difference $|f(x, y, z) - \widetilde{f}(x, y, z)|$ is bounded by a given value Δ – i.e., that the actual value $f(x, y, z)$ belongs to the interval

$$[\widetilde{f}(x, y, z) - \Delta, \widetilde{f}(x, y, z) + \Delta].$$

In this paper, we describe new efficient techniques for data compression under such interval uncertainty.

1 Formulation of the Problem

Compression Is Important. At present, so much data is coming from measuring instruments that it is necessary to compress this data before storing and processing. We can gain some storage space by using lossless compression, but often, this gain is not sufficient, so we must use lossy compression as well.

Successes of 2-D Image Compression. In the last decades, there has been a great progress in image and data compression. In particular, the JPEG2000 standard (see, e.g., [8]) uses the wavelet transform methods together with other efficient compression techniques to provide a very efficient compression of 2D images.

Within this standard, we can select different bitrates (i.e., number of bits per pixel that is required, on average, for the compressed image), and depending on the bitrate, get different degrees of compression.

When we select the highest possible bitrate, we get the lossless compressions that enables us to reconstruct the original image precisely. When we decrease the bitrate, we get a lossy compression; the smaller the bitrate, the more the compressed/decompressed image will differ from the original image.

2-D Data Compression. In principle, it is possible to use these compression techniques to compress 2D measurement data as well.

Compressing 3-D Data: Layer-by-Layer Approach. It is also possible to compress 3D measurement data $f(x, y, z)$ – e.g., meteorological measurements taken in different places (x, y) at different heights z.

One possibility is simply to apply the 2D JPEG2000 compression to each horizontal layer $f(x, y, z_0)$.

Compressing 3-D Data – An Approach That Uses KLT Transform: General Idea. Another possibility, in accordance with Part 2 of JPEG2000 standard, is to first apply the KLT transform to each vertical line. Specifically, we:

- compute the average value $\bar{f}(z) = N^{-1} \cdot \sum\limits_{x,y} f(x, y, z)$ of the analyzed quantity at a given height z, where N is the overall number of horizontal points (x, y);
- compute the covariances between different heights:

$$V(z_1, z_2) = N^{-1} \cdot \sum_{x,y} (f(x, y, z_1) - \bar{f}(z_1)) \cdot (f(x, y, z_2) - \bar{f}(z_2));$$

- find the eigenvalues λ_k and the eigenvectors $e_k(z)$ of the covariance matrix $V(z_1, z_2)$; we sort these eigenvectors into a sequence $e_1(z), e_2(z), \ldots$ so that $|\lambda_1| \geq |\lambda_2| \geq \ldots$;
- finally, we represent the original 3D data values $f(x, y, z)$ as a linear combination

$$f(x, y, z) = \bar{f}(z) + \sum_k a_k(x, y) \cdot e_k(z)$$

of the eigenvectors $e_k(z)$.

An Approach That Uses KLT Transform: Details. How can we implement this approach?

- The first two steps are straightforward.
- The third step – computing eigenvalues and eigenvectors – is a known computational problem for which many standard software packages provides an efficient algorithmic solution.
- So, to specify how this method can be implemented, we must describe how the last step can be implemented, i.e., how we can represent the original 3D data as a linear combination of the eigenvectors.

First, why is such a representation possible at all? It is known (see, e.g., [8]), that in general, the eigenvalues of a covariance matrix form a orthonormal basis in the space of all N_z-dimensional vectors $e = \{e(z)\} = (e(1), e(2), \ldots, e(N_z))$. By definitionof a basis this means, in particular, for every (x,y), the corresponding difference vector $d_{x,y}(z) \stackrel{\text{def}}{=} f(x,y,z) - \bar{f}(z)$ can be represented as a linear combination of these eigenvalues, i.e., that for every z, we have

$$d_{x,y}(z) = f(x,y,z) - \bar{f}(z) = \sum_k a_k(x,y) \cdot e_k(z),$$

where by $a_k(x,y)$, we denoted the coefficient at $e_k(z)$ in the corresponding expansion. From this formula, we get the desired representation of $f(x,y,z)$.

How can we actually compute this representation? Since the vectors $e_k(z)$ form an orthonormal basis, we can conclude that for the expansion of the vector $d_{x,y}$, each coefficient $a_k(x,y)$ at $e_k(z)$ is a dot (scalar) product of the vector $d_{x,y}$ and the k-th vector $e_k(z)$ from the basis, i.e., that

$$a_k(x,y) = d_{x,y} \cdot e_k \stackrel{\text{def}}{=} \sum_z d_{x,y}(z) \cdot e_k(z).$$

Substituting the expression for $d_{x,y}(z)$ into this formula, we conclude that

$$a_k(x,y) = \sum_z (f(x,y,z) - \bar{f}(z)) \cdot e_k(z).$$

Comment. Actually, instead of using this explicit (thus fast-to-compute) formula, we can use even faster formulas of the so-called *fast KLT transform* (see, e.g., [8]).

KLT Transform: Result. As a result of the KLT approach, we represent the original 3-D data as a sequence of the horizontal *slices* $a_k(x,y)$:

- the first slice $a_1(x,y)$ corresponds to the main (1-st) eigenvalue;
- the second slice $a_2(x,y)$ corresponds to the next (2-nd) eigenvalue;
- etc.,

with the overall intensity decreasing from one slice to the next one.

Next, to each of these slices $a_k(x,y)$, we apply a 2D JPEG2000 compression with the appropriate bit rate b_k depending on the slice k.

Decompressing 3-D Data: KLT-Based Approach. To reconstruct the data, we so the following:

- First, we apply JPEG2000 decompression to each slice; as a result, we get the values $\widetilde{a}_k^{[b_k]}(x,y)$.
- Second, based on these reconstructed slices, we now reconstruct the original 3-D data data as $\widetilde{f}(x,y,z) = \bar{f}(z) + \sum\limits_k \widetilde{a}_k^{[b_k]}(x,y) \cdot e_k(z)$.

This Approach Is Tailored Towards Image Processing – And Towards Mean Square Error. The problem with this approach is that for compressing measurement data, we use image compression techniques. The main objective of image compression is to retain the quality of the image. From the viewpoint of visual image quality, the image distortion can be reasonably well described by the mean square difference MSE (a.k.a. L^2-norm) between the original image $I(x, y)$ and the compressed-decompressed image $\widetilde{I}(x, y)$. As a result, sometimes, under the L^2-optimal compression, an image may be vastly distorted at some points (x, y) – and this is OK as long as the overall mean square error is small.

For Data Compression, MSE May Be a Bad Criterion. When we compress measurement results, however, our objective is to be able to reproduce each individual measurement result with a certain guaranteed accuracy.

In such a case, reconstruction that only guaranteed mean square error over the data set is unacceptable: for example, if we use the meteorological data to plan a best trajectory for a plane, what we really want to know are the meteorological parameters such as wind, temperature, and pressure along the trajectory.

If along this line, the values are not reconstructed accurately enough, the plane may crash – and the fact that on average, we get a good reconstruction, does not help.

An Appropriate Criterion for Data Compression. What we need is a compression that guarantees the given accuracy for all pixels, i.e., that guarantees that the L^∞-norm $\max\limits_{x,y,z} |f(x, y, z) - \widetilde{f}(x, y, z)|$ is small.

What We Need Is Data Compression Under Interval Uncertainty. In other words, what we need is a compression that guarantees that for each (x, y), the difference $|f(x, y, z) - \widetilde{f}(x, y, z)|$ is bounded by a given value Δ – i.e., that the actual value $f(x, y, z)$ belongs to the interval $[\widetilde{f}(x, y, z) - \Delta, \widetilde{f}(x, y, z) + \Delta]$.

Comment. In engineering applications of interval computations, "interval uncertainty" usually means that the problem's parameters are uncertain *parameters*, known only with interval uncertainty.

In the above data compression problem, we have a *non-parametric* problem, so the traditional engineering meaning of interval uncertainty does not apply. In our problem, by interval uncertainty, we mean guaranteed bounds on the loss of accuracy due to compression.

Need for Optimal Data Compression Under Interval Uncertainty. There exist several compressions that provide such a guarantee. For example, if for each slice, we use the largest possible bitrate – corresponding to lossless compression – then $\widetilde{a}_k(x, y) = a_k(x, y)$ hence $\widetilde{f}(x, y, z) = f(x, y, z)$ – i.e., there is no distortion at all.

What we really want is, among all possible compression schemes that guarantee the given upper bound Δ on the compression/decompression error, to find the scheme for which the average bitrate $\overline{b} \overset{\text{def}}{=} (1/N_z) \cdot \sum\limits_k b_k$ is the smallest possible.

In some cases, the bandwidth is limited, i.e., we know the largest possible average bitrate b_0. In such cases, among all compression schemes with $\overline{b} \le b_0$, we must find a one for which the L^∞ compression/decompression error is the smallest possible.

What We Have Done. In this paper, we describe new efficient (suboptimal) techniques for data compression under such interval uncertainty.

2 New Technique: Main Ideas, Description, Results

What Exactly We Do. Specifically, we have developed a new algorithm that uses JPEG2000 to compress 3D measurement data with guaranteed accuracy. We are following the general idea of Part 2 of JPEG2000 standard; our main contribution is designing an algorithm that selects bitrates leading to a minimization of L^∞ norm as opposed to the usual L^2-norm.

Let Us Start Our Analysis with a 2-D Case. Before we describe how to compress 3-D data, let us consider a simpler case of compressing 2-D data $f(x,y)$. In this case, for each bitrate b, we can apply the JPEG2000 compression algorithm corresponding to this bitrate value. After compressing/decompressing the 2-D data, we get the values $\widetilde{f}^{[b]}(x,y)$ which are, in general, slightly different from the original values $f(x,y)$.

In the interval approach, we are interested in the L^∞ error

$$D(b) \stackrel{\text{def}}{=} \max_{x,y} \left| \widetilde{f}^{[b]}(x,y) - f(x,y) \right| .$$

The larger the bitrate b, the smaller the error $D(b)$. When the bitrate is high enough – so high that we can transmit all the data without any compression – the error $D(b)$ becomes 0.

Our objective is to find the smallest value b for which the L^∞ error $D(b)$ does not exceed the given threshold Δ. For the 2-D case, we can find this optimal b_{opt} by using the following iterative *bisection* algorithm. In the beginning, we know that the desired bitrate lies between 0 and the bitrate B corresponding to lossless compression; in other words, we know that $b \in [b^-, b^+]$, where $b^- = 0$ and $b^+ = B$.

On each iteration of the bisection algorithm, we start with an interval $[b^-, b^+]$ that contains b_{opt} and produce a new half-size interval still contains b_{opt}. Specifically, we take a midpoint $b_{\text{mid}} \stackrel{\text{def}}{=} (b^- + b^+)/2$, apply the JPEG2000 compression with this bitrate, and estimate the corresponding value $D(b_{\text{mid}})$. Then:

- If $D(b_{\text{mid}}) \le \Delta$, this means that $b_{\text{opt}} \le b_{\text{mid}}$, so we can replace the original interval $[b^-, b^+]$ with the half-size interval $[b^-, b_{\text{mid}}]$.
- If $D(b_{\text{mid}}) > \Delta$, this means that $b_{\text{opt}} > b_{\text{mid}}$, so we can replace the original interval $[b^-, b^+]$ with the half-size interval $[b_{\text{mid}}, b^+]$.

After each iteration, the size of the interval halves. Thus, after k iterations, we can determine b_{opt} with accuracy 2^{-k}.

3-D Problem Is Difficult. In the 3-D case, we want to find the bitrate allocation $b_1, \ldots,$ b_{N_z} that lead to the smallest average bit rate b among all the allocations that fit within the given interval, i.e., for which the L^∞ compression/decompression error does not exceed the given value Δ: $D(b_1, b_2, \ldots) \le \Delta$.

For each bitrate allocation, we can explicitly compute this error, but there are no analytic formulas that describe this dependence, so we end up having to optimize a complex function with a large number of variables b_i.

Such an optimization is known to be a very difficult task, because the computational complexity of most existing optimization algorithms grows exponentially with the number of variables. There are theoretical results showing that in general, this growth may be inevitable; to be more precise, this problem is known to be NP-hard; see, e.g., [9].

The Source of Our Main Idea: Use of Enclosures in Interval Computations. To solve our problem, we use the experience of interval computations; see, e.g., [4].

In many areas of science and engineering, we are interested in the value of a physical quantity y that is difficult (or even impossible) to measure directly. To measure such quantities, we find auxiliary easier-to-measure quantities $x_1, \ldots, x_n$ that are related to y by a known algorithm $y = g(x_1, \ldots, x_n)$ – an algorithm that, in general, computes a complex functions with a large number of variables. Then, we measure x_i, and apply the algorithm f to the results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of measuring x_i. As a result, we get an estimate $\widetilde{y} = g(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ for y.

Since the measured values $\widetilde{x}_i$ are, in general, slightly different from the actual values x_i, a natural question is: what is the error of the resulting estimate?

In many practical situations, the only information that we have about the measurement errors $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$ of direct measurements is the upper bounds Δ_i on $|\Delta x_i|$ guaranteed by the manufacturer of the measuring instrument. In such situations, the only information that we have about the actual value x_i is that x_i lies in the interval $\mathbf{x}_i \stackrel{\text{def}}{=} [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$. In this case, the only information that we have about y is that y belongs to the range

$$\mathbf{y} = g(\mathbf{x}_1, \ldots, \mathbf{x}_n) \stackrel{\text{def}}{=} \{g(x_1, \ldots, x_n) \,|\, x_1 \in \mathbf{x}_1 \,\&\, \ldots \,\&\, x_n \in \mathbf{x}_n\}.$$

It is known that computing this range exactly is an NP-hard problem; see, e.g., [5]. Crudely speaking, NP-hard means that we cannot have an algorithm that always finished in reasonable time and that always produces the exact range.

The objective of interval computation is find *guaranteed* bounds for the actual value of y. Since we cannot find the exact range $\mathbf{y}$, researchers in interval computations design algorithms that provide us with an *enclosure* $\mathbf{Y} \supseteq \mathbf{y}$ for the actual range.

Our Main Idea: Using the Upper Estimate (Enclosure) for the Optimized Error Function. In our case, the problem is, e.g., to find, among all bitrate allocations $(b_1, b_2, \ldots)$ with $\overline{b} \le b_0$, the one for which the L^∞ compression/decompression error $D(b_1, b_2, \ldots)$ is the smallest possible.

Since it is difficult to minimize the original function $D(b_1, \ldots)$, we find easier-to-optimize upper estimate $\widetilde{D}(b_1, b_2, \ldots) \ge D(b_1, b_2, \ldots)$ and then find the values

b_i that minimize $\widetilde{D}(b_1,\ldots)$. As a result, we find an allocation b_i guaranteeing that $\widetilde{D}(b_1,\ldots) \le \widetilde{D}_{\min}$ and thus, that $D(b_1,\ldots) \le \widetilde{D}_{\min}$.

Since, in general, $D(b_1,\ldots) \le \widetilde{D}(b_1,\ldots)$, the resulting allocation is only *suboptimal* with respect to $D(b_1,\ldots)$.

Explicit Formula for the Enclosure. Since we use the KLT, the difference

$$f(x,y,z) - \widetilde{f}(x,y,z)$$

is equal to $\sum\limits_k (a_k(x,y) - \widetilde{a}_k^{[b_k]}(x,y)) \cdot e_k(z)$. Therefore, once we know the L^∞-norms $D_k(b_k) \stackrel{\text{def}}{=} \max\limits_{x,y} |a_k(x,y) - \widetilde{a}_k^{[b_k]}(x,y)|$ of the compression/decompression errors of each slice, we can conclude that $|a_k(x,y) - \widetilde{a}_k^{[b_k]}(x,y)| \le D_k(b_k)$, hence, that

$$|(a_k(x,y) - \widetilde{a}_k^{[b_k]}(x,y)) \cdot e_k(z)| \le D_k(b_k) \cdot E_k,$$

where $E_k \stackrel{\text{def}}{=} \max\limits_z |e_k(z)|$. Thus, the desired L^∞ error is bounded by

$$\widetilde{D}(b_1,\ldots) \stackrel{\text{def}}{=} \sum_k D_k(b_k) \cdot E_k,$$

Resulting Algorithm: Derivation. In accordance with the above idea, to get the (suboptimal) bitrate allocation b_i, we must minimize the function $\widetilde{D}(b_1,\ldots) = \sum\limits_k D_k(b_k) \cdot E_k$ under the condition that the $\sum\limits_k b_k = N_z \cdot b_0$. By using the Kuhn-Tucker approach, we can reduce this problem to the unconstrained problem – of finding stationary points of the function $\sum\limits_k D_k(b_k) \cdot E_k + \lambda \cdot \sum\limits_k b_k - N_z \cdot b_0$. By definition of a stationary point, derivatives w.r.t. all the variables b_i should be 0s, so we end up with the equation $-D_k'(b_k) = \lambda / E_k$, where the parameters λ should be selected based on the value b_0.

It can be easily shown that the other problem – of minimizing the average bitrate under the constraint that the compression/decompression error does not exceed Δ – leads to the same equation.

As we have mentioned, the function $D_k(b)$ decreases when b increases, so $D_k'(b) < 0$, with $D_k'(b) \to 0$ as b grows. It is therefore reasonable to represent the desired equation as $|D_k'(b_k)| = \lambda / E_k$.

What are the bounds on λ? The larger b_k, the smaller λ.

From the above formula, we conclude that $\lambda = |D_k'(b_k)| \cdot E_k$, hence $\lambda \le \Lambda_k \stackrel{\text{def}}{=} (\max\limits_b |D_k'(b)|) \cdot E_k$, so $\lambda \le \Lambda \stackrel{\text{def}}{=} \min\limits_k \Lambda_k$.

Algorithm: Description. Once we know, for each slice k, the dependence $D_k(b)$ of the corresponding L^∞-error on the bitrate b, we can find the desired (suboptimal) values b_k as follows.

At first, we compute the above-described values Λ_k and Λ. We know that $\lambda \le [\lambda^-, \lambda^+] := [0, \Lambda]$. We use bisection to sequentially halve the interval containing λ and eventually, find the optimal value λ.

Once we know an interval $[\lambda^-, \lambda^+]$ that contains λ, we pick its midpoint λ_{mid}, and then use bisection to find, for each k, the value b_k for which $|D_k'(b_k)| = \lambda_{\text{mid}} / E_k$. Based on these b_k, we compute the average bitrate $\overline{b}$. If $\overline{b} > b_0$, this means that we have chosen too small $\lambda_{\rm mid}$, so we replace the original λ-interval with a half-size interval $[\lambda_{\rm mid}, \lambda^+]$. Similarly, if $\overline{b} < b_0$, we replace the original λ-interval with a half-size interval $[\lambda^-, \lambda_{\rm mid}]$.

After k iterations, we get λ with the accuracy 2^{-k}, so a few iterations are sufficient. Once we are done, the values b_k corresponding to the final $\lambda_{\rm mid}$ are returned as the desired bitrates.

The only remaining question is how to determine the dependence $D_k(b)$. In principle, we can try, for each layer k, all possible bitrates b, and thus get an empirical dependence $D_k(b)$.

We have shown, that this dependence can be described by the following analytical formula: $A_1 \cdot (b - b_0)^\alpha$ for all the values b until a certain threshold b_0, and $A_2 \cdot 2^{-b}$ for $b \geq b_0$. This model is a slight modification of a model from [6]. So, instead of trying all possible values b, it is sufficient to try a few to determine the values of the parameters A_i, b_0, and α corresponding to the given layer.

Results. To test our algorithm, we applied it to 3-D meteorological data: temperature T, pressure P, the components U, V, and W of the wind speed vector, and the waver vapor missing ratio WV.

For meteorological data, the resulting compression indeed leads to a much smaller L^∞ error bound $\Delta_{\rm new}$ than the L^∞ error bound $\Delta_{\rm MSE}$ corresponding to the bitrate allocation that optimizes the MSE error. The ratio $\Delta_{\rm new}/\Delta_{\rm MSE}$ decreases from 0.7 for $b_0 = 0.1$ to 0.5 for $b_0 = 0.5$ to ≤ 0.1 for $b_0 \geq 1$.

Acknowledgments

This work was partially supported by NSF Grant CDA-9522207, the ARO grants DAA H04-95-1-0494 and DAA D19-99-1-0012, by the Texas Instruments Grant, and by NASA under cooperative agreement NCC5-209. This work was partly performed during O. Kosheleva's visit to Brazil; this visit was sponsored by the Brazilian funding agency CTINFO/CNPq.

The authors are thankful to the anonymous referees for valuable suggestions.

References

1. Cabrera, S.D.: Three-Dimensional Compression of Mesoscale Meteorological Data based on JPEG2000, Battlespace Digitization and Network-Centric Warfare II, Proc. SPIE **4741** (2002) 239–250.
2. Kosheleva, O.M.: Task-Specific Metrics and Optimized Rate Allocation Applied to Part 2 of JPEG2000 and 3-D Meteorological Data, Ph.D. Dissertation, University of Texas at El Paso, 2003.
3. Kosheleva, O.M., Usevitch, B., Cabrera, S., Vidal, E.Jr.: MSE Optimal Bit Allocation in the Application of JPEG2000 Part 2 to Meteorological Data, Proceedings of the 2004 IEEE Data Compression Conference DCC'2004, Snowbird, Utah, March 23–25, 2004, p. 546.
4. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis, Springer, London, 2001.

5. Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: Computational Complexity and Feasibility of Data Processing and Interval Computations, Kluwer, Dordrecht, 1997.
6. Mallat, S., Falzon, F.: Analysis of low bit rate image transform coding, IEEE Trans. Signal Proc. **46** (1998) 1027–1042.
7. Moore, R.E.: Methods and Applications of Interval Analysis. SIAM, Philadelphia, 1979.
8. Taubman, D.S., Marcellin, M.W.: JPEG2000 Image Compression Fundamentals, Standards and Practice, Kluwer, Boston, Dordrecht, London, 2002.
9. Vavasis, S.A.: Nonlinear Optimization: Complexity Issues, Oxford University Press, New York, 1991.

Computing Interval Bounds for Statistical Characteristics Under Expert-Provided Bounds on Probability Density Functions

Victor G. Krymsky

Ufa State Aviation Technical University
12 K. Marx Street, Ufa, Bashkortostan, 450000, Russia
kvg@mail.rb.ru

Abstract. The paper outlines a new approach developed in the framework of imprecise prevision theory. This approach uses expert-provided bounds as constraints on the values of probability density functions. Such approach allows us to overcome the difficulties caused by using traditional imprecise reasoning technique: by eliminating non-physical degenerate distributions, we reduce the widths of the resulting interval estimates and thus, make these estimates more practically useful.

1 Introduction

Imprecise prevision theory (IPT) started from fundamental publications [1,2] (see also [3]); it now attracts significant attention of researchers from many countries. An important advantage of IPT is its capacity to combine both objective statistical information and expert information when estimating the lower and the upper bounds on probabilities and other relevant characteristics. Such estimates can be obtained without introducing any additional assumptions about a probability distribution; these estimates are usually obtained by solving problems of linear programming type.

In some practical applications, however, the bounds generated by the traditional IPT approach are too wide, so wide that they are not very practically useful. This happens, e.g., in practical problems related to reliability and risk analysis. As shown in [4], one of the main reasons why these bounds are too wide is exactly that in the IPT methodology, we do not make any assumption about the probability distribution. As a result, the "optimal" distributions, i.e., the distributions for which the estimated characteristic attains its largest possible (or smallest possible) value, are degenerate – in the sense that the probability distribution is located on a (small) finite set of values (and the corresponding probability density function is a linear combination of the δ-functions). Such degenerate distributions are not physically possible – e.g., it is not possible that the failure occurs only at certain moments of time. It is therefore desirable to restrict ourselves to physically possible probability distributions; hopefully, such a restriction would decrease the width of the resulting interval estimates and thus, make these estimates more practically useful.

A natural way to impose such a restriction is when we know an upper bound on the values of the probability density. In many practical situations, such upper bounds can be elicited from experts. For example, in reliability problems, we can ask an expert questions like “what is the largest possible percentage of failures per year for a given component”. In other cases, such bounds can be obtained from the statistical data or from the physical models of the corresponding phenomena.

It is worth mentioning that in all these cases, all we get is an upper bound on the value of the probability density, we do not get any information about the actual shape of the corresponding probability distribution.

Once we know such an upper bound, we can use it to restrict possible probability distributions. In other words, we can use these upper bounds as additional constraints in the original optimization problems; as we will show, we will then be able to use variational calculus to solve the corresponding optimization problems and thus, generate the desired narrower interval estimates for the statistical characteristics which are of interest to us.

2 Traditional Formulation of the Problem

In order to describe how we can use the upper bounds on the probability density, let us first recall how the corresponding problems are formulated and solved in the situation when no such bounds are available.

Let us restrict ourselves to the practically useful one-dimensional case, in which we are interested in the distribution of a single (real-valued) variable. Formally, let $\rho(x)$ be an (unknown) probability density function of a random variable X distributed on the interval $[0, T]$. In the traditional formulation of an IPT problem for one-dimensional case (see, e.g., [1,2,3,4]), we assume that we know the bounds $\underline{a}_i$ and $\overline{a}_i$ on the (unknown) expected values $a_i = M(f_i)$ of several characteristics $f_i(x)$, i.e., functions of this random variable. For example, if $f_i(x) = x^k$, then the expected value of $f_i(x)$ is the corresponding moment; if $f_i(x) = 1$ for $x \le x_0$ and 0 otherwise, then the expected value is the probability that $x \le x_0$, etc.

The class of all probability distributions that are consistent with our knowledge about $\rho(x)$ can be therefore described by the following constraints:

$$\rho(x) \ge 0, \int_0^T \rho(x)\,dx = 1, \text{ and } \underline{a_i} \le \int_0^T f_i(x) \cdot \rho(x)\,dx \le \overline{a_i}, \quad i = 1, 2, \ldots, n. \quad (1)$$

Here, $f_i(x)$ are given real-valued functions; in most cases, these functions are non-negative. (In the IPT approach [1,2,3], such functions are called *gambles*.) The values $\underline{a}_i, \overline{a_i} \in R_+$ are given numbers.

We are interested in the expected value $M(g)$ of an additional non-negative *gamble* $g(x)$. For different distributions satisfying the constraint (1), we may get different expected values of $g(x)$. What we are therefore interested in is the *interval* of possible values of $M(g)$. The smallest possible value of $M(g)$ is called the *coherent lower prevision* and denoted by $\underline{M}(g)$; the largest possible value of $M(g)$ is called the *coherent upper prevision* and denoted by $\overline{M}(g)$; Thus, computing the coherent lower and upper previsions $\underline{M}(g)$ and $\overline{M}(g)$ for the expectation $M(g)$ of the function $g(x)$ means estimating

$$\inf_{\rho(x)} \int_0^T g(x) \cdot \rho(x)\, dx, \text{ and } \sup_{\rho(x)} \int_0^T g(x) \cdot \rho(x)\, dx, \tag{2}$$

subject to the constraints (1).

Both the objective function (2) and the constraints (1) are linear in terms of the values of the unknown function $\rho(x)$. Thus, the optimization problem (1)–(2) requires that we optimize a linear function under linear equalities and inequalities; in other words, the optimization problem (1)–(2) is a linear programming-type problem.

One of the known ways to solve linear programming problems is to form *dual* linear programming problems. It is known [1,2,3] that in many practically useful cases, the dual problem can be easily solved – thus, the original problem can be solved as well.

For the optimization problem (1)–(2), the dual problem has the following form:

$$\underline{M}(g) = \sup_{c_0, c_i, d_i} \left(c_0 + \sum_{i=1}^{n} (c_i \cdot \underline{a}_i - d_i \cdot \overline{a}_i) \right), \tag{3}$$

where the supremum is taken over all $c_0 \in R$ and $c_i, d_i \in R_+$ for which, for every $x \geq 0$, we have

$$c_0 + \sum_{i=1}^{n} (c_i - d_i) \cdot f_i(x) \leq g(x). \tag{4}$$

Similarly,

$$\overline{M}(g) = \inf_{c_0, c_i, d_i} \left(c_0 + \sum_{i=1}^{n} (c_i \cdot \overline{a}_i - d_i \cdot \underline{a}_i) \right), \tag{5}$$

where the supremum is taken over all $c_0 \in R$ and $c_i, d_i \in R_+$ for which, for every $x \geq 0$, we have

$$c_0 + \sum_{i=1}^{n} (c_i - d_i) \cdot f_i(x) \geq g(x). \tag{6}$$

In [4], it was shown that, in general, the functions $\rho(x)$ at which $M(g)$ attains its maximum and its minimum – i.e., at which it attains the values $\underline{M}(g)$ and $\underline{M}(g)$ – are degenerate – i.e., probability density functions which are linear combinations of δ-functions.

The fact that we get such physically unrealistic (and hence, undesirable) probability distributions is due to the lack of restrictions of $\rho(x)$. That such distributions are allowed shows that in the traditional approach, the level of uncertainty is unreasonably high.

It is therefore desirable to dis-allow such physically unreasonable probability distributions – by explicitly introducing upper bounds on the probability density. This approach is described, in detail, in the next section.

3 Modified Formulation of the Problem: The Case of Bounded Densities

As we have mentioned, often, in addition to the restrictions (1), we also know an upper bound $K \in R_+$ on the values of the probability density function, i.e., we know that for every x:

$$\rho(x) \leq K = \text{const.} \tag{7}$$

Comment. Since the overall probability on the interval $[0, T]$ has to be equal to 1, the upper bound K must satisfy the inequality $K \cdot T \geq 1$.

In the new formulation of the problem, we must find the optimal values of the objective function (2) subject to constraints (1), (7).

How can we solve this new problem? This new optimization problem is still a problem of optimizing a linear function (2) under linear constraints (1) and (7). However, in this new problem, we have infinitely many variables and *infinitely many constraints*, so, to solve this problem, we can no longer apply the dual problem approach, an approach that is so helpful in the solution of the traditional problem (1)–(2). Indeed, the number of variables in the dual problem is equal to the number of constraints in the original problem. In the formulation (1)–(2), we had a problem with a small number of constraints (1). In a dual problem (3)–(4) (or (5)–(6)), we, accordingly, had a small number of variables; so, although we had infinitely many constraints, the problem was still much easier to solve than the original problem (1)–(2). In our new problem, we have infinitely many constraints (7); so, the dual problem also has infinitely many variables (and infinitely many constraints). Hence, for the new problem, duality does not help.

To solve the new problems, we propose a new idea based on the following theorem:

Theorem. *If there is no non-degenerate interval $[\alpha, \beta] \subseteq [0, T]$ on which the function $g(x)$ can be represented in the form*

$$g(x) = h_0 + \sum_{i=1}^{n} h_i \cdot f_i(x), \tag{8}$$

for some $h_0, h_1, \ldots, h_n \in R$, then each piece-wise continuous function $\rho(x)$ providing the solution to the above optimization problem mentioned (1), (2), (7), is a piece-wise constant step-function whose values are either 0 or K.

Comment. In practice, we can restrict ourselves to piece-wise continuous functions $\rho(x)$.

Comment. In many practical situations, the functions $f_i(x)$ and $g(x)$ are smooth (differentiable). For smooth functions, the equality (8) takes place only if the system of linear algebraic equations

$$h_0 + \sum_{i=1}^{n} h_i \cdot f_i(x) = g(x);\ \sum_{i=1}^{n} h_i \cdot f_i'(x) = g'(x); \cdots \sum_{n=1}^{n} h_i \cdot f_i^{(n)}(x) = g^{(n)}(x), \tag{9}$$

with unknowns $h_0, h_1, \ldots, h_n$ has at least one solution (independent on x) on some interval $x \in [\alpha, \beta]$. (Here we used the standard notations for the derivatives: $g^{(l)}(x) = d^{(l)}g(x)/dx^{(l)}$; $f_i^{(l)}(x) = d^{(l)}f_i(x)/dx^{(l)}$, $i, l = 1, 2, \ldots, n$.)

In most practical situations, this condition is not satisfied – so, the theorem is applicable in most practical situations.

Proof. Let us apply the methods of variational calculus to the above optimization problem, with the constraints (1), (7). To be able to do that, we must somehow eliminate the inequalities

$$0 \le \rho(x) \le K \tag{10}$$

because the variational calculus is based on the idea that we can always apply a small change $\rho(x) \to \rho(x)+\delta\rho(x)$ to an unknown function – i.e., in topological terms, that the set of possible functions $\rho(x)$ is an open domain. On the other hand, with the inequalities (10), when $\rho(x) = 0$ or $\rho(x) = K$, a small change in $\rho(x)$ leads us outside the domain.

The requirement $\rho(x) \ge 0$ can be eliminated by introducing a new unknown function $z(x)$ for which

$$\rho(x) = z^2(x). \tag{11}$$

(We then have to substitute $z^2(x)$ instead of $\rho(x)$ in the expressions for objective functions and for the constraints.)

To take the requirement (7) into account, we use an idea from the optimal control theory (see, e.g., [5]) and introduce yet another unknown function $v(x)$ with the additional restriction that

$$z^2(x) + v^2(x) = K. \tag{12}$$

Now, we must optimize w.r.t. both unknown functions $z(x)$ and $v(x)$. In terms of these new unknown functions, the original problem takes the following form:

We want to find

$$\inf_{z(x)} \int\limits_0^T g(x) \cdot z^2(x)\, dx \text{ and } \sup_{z(x)} \int\limits_0^T g(x) \cdot z^2(x)\, dx \tag{13}$$

subject to

$$z^2(x) + v^2(x) = K, \tag{14}$$

$$\int\limits_0^T z^2(x)\, dx = 1, \quad \int\limits_0^T f_i(x) \cdot z^2(x)\, dx \le \overline{a_i}, \quad -\int\limits_0^T f_i(x) \cdot z^2(x)\, dx \le -\underline{a_i}, \tag{15}$$

where $i = 1, 2, \ldots, n$.

In variational calculus, expressions like (14) (algebraic relations between the unknown functions) are called *holonomic constraints*, and equalities and inequalities like (15) – that estimate the values of some integrals depending on the unknown functions – are called belong *isoperimetric constraints*. The standard way of solving the corresponding optimization problem is to replace the original constraint optimization problem with the objective function

$$F(z, v) = g(x) \cdot z^2(x) \tag{16}$$

by an unconstrained optimization problem with a new objective function

$$\begin{aligned} F^*(z, v) &= g(x) \cdot z^2(x) + \lambda^*(x) \cdot (z^2(x) + v^2(x)) + \lambda_0 \cdot z^2(x) \\ &\quad + \sum_{i=1}^{n} \lambda_i \cdot f_i(x) \cdot z^2(x) - \sum_{i=n+1}^{2n} \lambda_i \cdot f_i(x) \cdot z^2(x) \\ &= g(x) \cdot z^2(x) + \lambda^*(x) \cdot (z^2(x) + v^2(x)) + \lambda_0 \cdot z^2(x) \\ &\quad + \sum_{i=1}^{n} (\lambda_i - \lambda_{i+n}) \cdot f_i(x) \cdot z^2(x), \end{aligned} \tag{17}$$

where $\lambda^*(x), \lambda_0, \lambda_1, \ldots, \lambda_{2n}$ are the (unknown) Lagrange multipliers.

Note that $\lambda^*(x)$ is a function of x because it corresponds to a holonomic constraint; on the other hand, $\lambda_0, \lambda_1, \ldots, \lambda_{2n}$ are constants because they correspond to isoperimetric constraints.

For unconstrained optimization problem, it is known that the Euler-Lagrange equations provide the necessary condition of optimality. For our problem, these equations take the following form:

$$\frac{\partial F^*(z,v)}{\partial z} = \frac{d}{dx}\left(\frac{\partial F^*(z,v)}{\partial \dot{z}}\right); \ \frac{\partial F^*(z,v)}{\partial v} = \frac{d}{dx}\left(\frac{\partial F^*(z,v)}{\partial \dot{v}}\right), \tag{18}$$

where $\dot{z} \overset{\text{def}}{=} dz/dx$; $\dot{v} \overset{\text{def}}{=} dv/dx$. For the function (17), these equations take the form:

$$z(x) \cdot \left(g(x) + \lambda^*(x) + \lambda_0 + \sum_{i=1}^{n} (\lambda_i - \lambda_{i+n}) f_i(x) \right) = 0; \tag{19}$$

$$\lambda^*(x) \cdot v(x) = 0. \tag{20}$$

The equations (19), (20) should be considered together with the constraints (14), (15).

Let us consider an arbitrary point $x_0 \in [0,T]$ at which $z(x_0) \neq 0$. Since the function $z(x)$ is assumed to be piece-wise continuous, it is either continuous to the right of x_0 or to the left of x_0. Without loosing generality, let us assume that it is continuous to the right of x_0, i.e., on some interval $[x_0, \beta]$ with $\beta > x_0$. Since the function $z(x)$ is continuous on this interval and $z(x_0) \neq 0$, we conclude that for some $\varepsilon > 0$, we have $z(x) \neq 0$ for all $x \in [x_0, x_0 + \varepsilon]$. For every $\delta \in (0, \varepsilon)$, from (19) and $z(x) \neq 0$, we conclude that for all $x \in [x_0, x_0 + \delta]$, we have

$$\lambda^*(x) = -g(x) - \lambda_0 - \sum_{i=1}^{n} (\lambda_i - \lambda_{i+n}) \cdot f_i(x).$$

Due to the main assumption of the theorem – that the condition (8) cannot take place everywhere on an interval – we have $\lambda^*(x) \neq 0$ for some $x_\delta \in [x_0, x_0 + \delta]$. For this x_δ, the equation (20) implies that $v(x_\delta) = 0$ – hence, due to (14), that $z(x_\delta) = \sqrt{K}$. When $\delta \to 0$, we have $x_\delta \to x_0$; since the function $z(x)$ is continuous on the interval $[x_0, x_0 + \beta]$, we conclude, in the limit, that $z(x) = \sqrt{K}$. So, if a piece-wise constant continuous function $z(x)$ attains a non-zero value for some x, this value can only be $\sqrt{K}$ which corresponds to $\rho(x) = K$. The theorem is proven.

This theorem enables us to reduce the original difficult-to-solve variational optimization problem to an easier-to-solve problem of optimizing a multivariate function under algebraic constraints.

Indeed, let (x_0, x_1), (x_2, x_3), $(x_4, x_5), \ldots, (x_{2m}, x_{2m+1})$ be the intervals on which $\rho(x) = K \neq 0$, and let $(x_1, x_2), \ldots$ be the intervals on which $\rho(x) = 0$ (Fig. 1).
Let us denote

$$G(x_j, x_{j+1}) \overset{\text{def}}{=} \int_{x_j}^{x_{j+1}} g(x)\,dx; \ \Phi_i(x_j, x_{j+1}) \overset{\text{def}}{=} \int_{x_j}^{x_{j+1}} f_i(x)\,dx, \ i = 1, 2, \ldots, n. \tag{21}$$

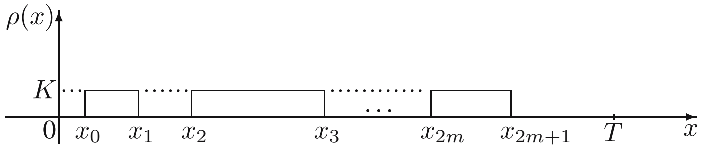


Fig. 1. The optimal density function

Then, the problem (1), (2), (7) takes the following form:

We want to find

$$\min_{x_0,x_1,\dots}\left\{K\cdot\sum_{j=0}^{m}G(x_{2j},x_{2j+1})\right\}\text{ and }\max_{x_0,x_1,\dots}\left\{K\cdot\sum_{j=0}^{m}G(x_{2j},x_{2j+1})\right\}\qquad(22)$$

subject to

$$K\cdot\sum_{j=0}^{m}(x_{2j+1}-x_{2j})=1;\ \underline{a_i}\le K\cdot\sum_{j=0}^{m}\varPhi(x_{2j},x_{2j+1})\le\overline{a_i},\ i=1,2,\dots,n.\qquad(23)$$

Once we know the number of intervals m, we can solve this optimization problem by using standard numerical techniques such as gradient methods, simplex-based search methods, genetic algorithms, etc. In simple situations, solution can be obtained in analytical form.

How can we find m? A natural idea is to start with a small value m (e.g., with $m=0$), to solve the optimization problem for this m, then increase m by 1, solve the problem again, etc. We stop if for the new m, we get the exact same optimizing function $\rho(x)$ as for the previous m – this means that a further subdivision of intervals will probably not improve the value of the objective function (2).

Example 1. Let us consider the simplest possible example in which the inequality (7) is the only restriction on $\rho(x)$ (i.e., $n=0$). In this case, what are the bounds on the expected value $M(x)$ of the corresponding random variable?

In this example, $g(x)=x$. Obviously, there exists no interval on which $g(x)=x=c_0=\text{const}$, so the theorem can be applied.

Let us start with the case $m=0$, i.e., with the case when the solution of the optimization problem is equal to K on a single interval (x_0,x_1). Here we have only one constraint $\int_0^T\rho(x)\,dx=K\cdot(x_1-x_0)=1$, hence $x_1=1/K+x_0$. The objective function takes the form

$$J=\int_0^T x\cdot\rho(x)\,dx=K\cdot\int_0^T x\,dx=K\cdot\frac{x_1^2-x_0^2}{2}.$$

Substituting $x_1=1/K+x_0$ into the expression for J, we conclude that $J=K\cdot\dfrac{2x_0/K+1/K^2}{2}=x_0+\dfrac{1}{2K}$. Here, $x_0\ge0$ and $x_1\le T$ – hence $x_0\le T-\dfrac{1}{2K}$.

It is clear that the smallest value of J is attained when $x_0 = 0$, so $\min J = \frac{1}{2K}$. Similar, to get $J \to \max$, we take the largest possible value of x_0, i.e., $x_0 = T - 1/K$, hence $\max J = \left(T - \frac{1}{K}\right) + \frac{1}{2K} = T - \frac{1}{2K}$.

Now, according to our procedure, we try $m = 1$. In this case, we have

$$J = \int_0^T x \cdot \rho(x)\,dx = K \cdot \left(\int_{x_0}^{x_1} x\,dx + \int_{x_2}^{x_3} x\,dx\right) = K \cdot \left(\frac{x_1^2 - x_0^2}{2} + \frac{x_3^2 - x_2^2}{2}\right).$$

The constraint becomes

$$\int_0^T \rho(x)\,dx = K \cdot (x_1 - x_0 + x_3 - x_2) = 1,\ x_0 \geq 0;\ x_1 \leq T.$$

It is easy to see that the requirement $J \to \min$ leads to $x_2 = x_0 = 0$ and $x_3 = x_1 = 1/K$; so, the new optimal function is identical to the function corresponding to $m = 0$. The similar situation occurs for maximizing J; we can show that the choice of $m = 0$ is really optimal.

Example 2. In this example, we are again interested in the bounds on the expected value, but we now have an additional constraint

$$\text{Prob}(x \in [\underline{q}, \overline{q}]) = \int_0^T f_1(x) \cdot \rho(x)\,dx = b,$$

for some $\underline{q}, \overline{q} \in R_+$. Here, $f_1(x) = I_{[\underline{q},\overline{q}]}(x)$ is the characteristic function of the interval, i.e., the function that is equal to 1 if $x \in [\underline{q}, \overline{q}]$ and to 0 otherwise.

Here the theorem can be also applied. Further analysis shows that $m = 1$ is the best choice for such situations. To provide $J \to \min$ we have to set: if $\underline{q} \geq (1 - b)/K$,

$$\rho(x) = \begin{cases} K \text{ for } 0 \leq x \leq (1-b)/K; \\ 0 \text{ for } (1-b)/K < x < \underline{q}; \\ K \text{ for } \underline{q} \leq x \leq \underline{q} + b/K; \\ 0 \text{ for } \underline{q} + b/K < x \leq T; \end{cases}$$

and for $\underline{q} < (1 - b)/K$:

$$\rho(x) = \begin{cases} K \text{ for } 0 \leq x \leq \underline{q} + b/K; \\ 0 \text{ for } \underline{q} + b/K < x < \overline{q}; \\ K \text{ for } \overline{q} \leq x \leq \overline{q} + (1 - \underline{q} \cdot K - b)/K; \\ 0 \text{ for } \overline{q} + (1 - \underline{q} \cdot K - b)/K < x \leq T. \end{cases}$$

As a result, we get, correspondingly,

$$\min J = \frac{1}{2K} + \frac{b \cdot (\underline{q} \cdot K - (1-b))}{K}$$

or

$$\min J = \frac{1}{2K} + \frac{\overline{q} \cdot K - (\underline{q} \cdot K + b)(1 + (\overline{q} - \underline{q}) \cdot K - b)}{K}.$$

Solution of the problem $J \to \max$ can be obtained in a similar way.

Let us compare this solution with the solution to the corresponding "traditional" IPT problem – in which we impose no restrictions on the upper bound of $\rho(x)$. In this case, the dual problem has the form $c_0 + c_1 \cdot b \to \max$ under the constraint $g(x) = x \geq c_0 + c_1 \cdot I_{[\underline{q},\overline{q}]}(x)$. To satisfy this constraint, we have to place the function $\psi(x) \overset{\text{def}}{=} c_0 + c_1 \cdot I_{[\underline{q},\overline{q}]}(x)$ under the curve $y = x$ (as shown in Fig. 2).

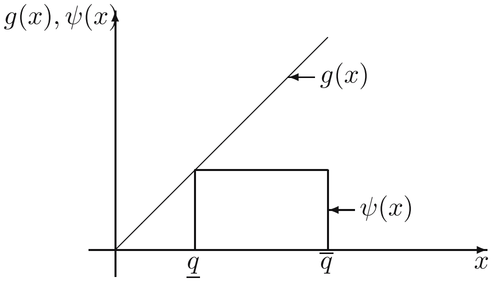


Fig. 2. Plots of the functions $g(x)$ and $\psi(x)$

Hence, we obtain $c_0 = 0$ and $c_1 = \underline{q}$, so $\underline{M}[g]_{trad} = \underline{q} \cdot b$. Let us find the difference $\Delta(\min J) = \min J - \underline{M}[g]_{trad}$.

If $\underline{q} \geq \dfrac{1-b}{K}$, then

$$\Delta(\min J) = \frac{1}{2K} + \frac{b \cdot (\underline{q} \cdot K - (1-b))}{K} - \underline{q} \cdot K = \frac{b^2 + (1-b)^2}{2K} > 0.$$

So, the expert-provided upper bound K on $\rho(x)$ indeed improves the bounds on J – and the smaller the bound the larger the improvement. For example, for $T = 1$, $K = 5.0$, $\underline{q} = 0.2$, and $b = 0.5$, we get $\underline{M}[g]_{trad} = 0.1$ and $\Delta(\min J) = 0.05$ – a 50% improvement.

If $\underline{q} < \dfrac{1-b}{K}$, then

$$\Delta(\min J) = \frac{(b-1)^2 + b^2 + 2b \cdot (\overline{q} - \underline{q}) \cdot (1 - b - \underline{q} \cdot K)}{2K} > 0.$$

Acknowledgments

I would like to express my sincere gratitude to Dr. Igor Kozine of Risø National Laboratory, Denmark, for kind attention to this research. I am very grateful to Professor Vladik Kreinovich of University of Texas at El Paso, the USA, for valuable suggestions, important comments and helpful corrections. The work was partially supported by the grant T02-3.2-346 of Russian Ministry for Education.

References

1. Walley, P. Statistical reasoning with imprecise probabilities. Chapman and Hall. New York, 1991.
2. Kuznetsov, V. Interval statistical models. Radio and Sviaz. Moscow, 1991 (in Russian).
3. Kuznetsov, V. Interval methods for processing statistical characteristics. Proceedings of APIC'95 Conference at El Paso, Extended Abstracts, A Supplement to Int. Journal of Reliable Computing, 1995, pp. 116–122.
4. Utkin, L., Kozine, I. Different faces of the natural extension. Proceedings of the Second International Symposium on Imprecise Probabilities and Their Applications, ISIPTA '01, 2001, pp. 316–323.
5. Ivanov, V.A., Faldin, N.V. Theory of optimal control systems. Nauka. Moscow, 1981 (in Russian).

Interval Parallel Global Optimization with Charm++

José A. Martínez, Leocadio G. Casado,
José A. Alvarez, and Inmaculada García

Computer Architecture and Electronics Dpt.
University of Almería
04120 Almería, Spain
{jamartin,leo,jaberme,inma}@ace.ual.es

Abstract. Interval Global Optimization based on Branch and Bound (B&B) technique is a standard for searching an optimal solution in the scope of continuous and discrete Global Optimization. It iteratively creates a search tree where each node represents a problem which is decomposed in several subproblems provided that a feasible solution can be found by solving this set of subproblems. The enormous computational power needed to solved most of the B&B Global Optimization problems and their high degree of parallelism make them suitable candidates to be solved in a multiprocessing environment. This work evaluates a parallel version of AMIGO (Advanced Multidimensional Interval Analysis Global Optimization) algorithm. AMIGO makes an efficient use of all the available information in continuous differentiable problems to reduce the search domain and to accelerate the search. Our parallel version takes advantage of the capabilities offered by Charm++. Preliminary results show our proposal as a good candidate to solve very hard global optimization problems.

1 Introduction

The problem of finding the global minimum f^* of a real valued n-dimensional continuously differentiable function $f : S \to \mathbb{R},\ S \subset \mathbb{R}^n$, and the corresponding set S^* of global minimizers is considered, i.e.:

$$f^* = f(s^*) = \min_{s \in S} f(s),\ s^* \in S^*. \tag{1.1}$$

The following notation is used. $\mathbb{I} = \{X = [a,b] \mid a \le b; a,b \in \mathbb{R}\}$ is the set of all one-dimensional intervals. $X = [\underline{x}, \overline{x}] \in \mathbb{I}$ is a one-dimensional interval. $X = (X_1, \ldots, X_n) \subseteq S, X_i \in \mathbb{I},\ i = 1, \ldots, n$ is an n-dimensional interval, also called box. $\mathbb{I}^n$ is the set of the n-dimensional intervals. $f(X) = \{f(x) \mid x \in X\}$ is the real range of f on $X \subseteq S$. F and $F' = (F'_1, \ldots, F'_n)$ are interval inclusion functions of f and its derivative f', respectively. These interval inclusion functions satisfy that: $f(X) \subseteq F(X)$ and $f'(X) \subseteq F'(X)$ [8].

In those cases where the objective function $f(x)$ is given by a formula, it is possible to use an interval analysis B&B approach to solve problem (1.1) (see [6,8,9,10]). A general interval GO (IGO) algorithm based on this approach is shown in Algorithm 1. An overview on theory and history of the rules of this algorithm can be found, for

J. Dongarra, K. Madsen, and J. Waśniewski (Eds.): PARA 2004, LNCS 3732, pp. 161–168, 2006.

Algorithm 1 : A general interval B&B GO algorithm

```
Funct IGO($S$, $f$)

 1. Set the working list $L$ := $\{S\}$ and the final list $Q$ := $\{\}$
 2. while ( $L \neq \{\}$ )
 3.   Select an interval $X$ from $L$                          Selection rule
 4.   Compute a lower bound of $f(X)$                          Bounding rule
 5.   if $X$ cannot be eliminated                              Elimination rule
 6.     Divide $X$ into $X^j$, $j = 1, \dots, p$, subintervals   Division rule
 7.     if $X^j$ satisfies the termination criterion           Termination rule
 8.       Store $X^j$ in $Q$
 9.     else
10.       Store $X^j$ in $L$
11. return Q
```

example, in [6]. Of course, every concrete realization of Algorithm 1 depends on the available information about the objective function $f(x)$. The interval global optimization algorithm used in this article is called AMIGO [7].

AMIGO supposes that interval inclusion functions can be evaluated for $f(x)$ and its first derivative $f'(x)$ on X. Thus, the information about the objective function which can be obtained during the search is:

$$F(x),\ F(X) \text{ and } F'(X). \tag{1.2}$$

When the information stated in (1.2) is available, the rules of a traditional realization of Algorithm 1 can be written more precisely. Below we shortly describe the main characteristics of AMIGO. A detailed description can be found in [7].

The Bounding rule lets to get a lower and upper bounds of $f(X)$. Interval arithmetic provides a natural and rigorous way to compute these bounds by changing the real function to its interval version $F(X)$. Better approximations are obtained using derivative information. The Selection rule chooses among all the intervals X^j stored in the working list L, the interval X with a better lower bound of $\underline{f}(X)$. Most of the research of interval global optimization algorithm was done in the elimination rule to reduce as much as possible the search space. AMIGO incorporates the traditional elimination rules (Cutoff and Monotonicity tests) and additionally can reduce the size of an interval using the information given in (1.2). Cutoff test: An interval X is rejected when $\underline{F}(X) > \overline{\tilde{f}}$, where $\overline{\tilde{f}}$ is the best known upper bound of f^*. The value of $\tilde{f} = [\underline{\tilde{f}}, \overline{\tilde{f}}]$ is usually updated by evaluating F at the middle point of X. Monotonicity test: If condition $0 \notin F'(X)$ for an interval X is fulfilled, then this means that the interval X does not contain any minimum and, therefore, can be rejected. The easier division rule usually generates two subintervals using bisection on the direction of the wider coordinate. The termination rule determines the desired accuracy of the problem solution. Therefore, intervals X with a width smaller or equal to a value ε, are moved to the final list Q.

This deterministic global optimization algorithm exhibits a strong computational cost, mainly for hard to solve problems. Nevertheless, it exhibits a high degree of parallelism which can be exploited by using a multicomputer system. It also exhibits a high degree of irregularity. This means that special attention has to be paid to the load balancing and communication cost problems.

In this work we are using a SPMD parallel programing model. In our proposals, the set of subproblems generated by the B&B procedure is distributed among processors following a random strategy, which is appropriate when the number of generated subproblems is huge.

This paper is outlined as follows: Section 2 describes some issues related to the general framework of parallel B&B algorithms. Section 3 is a brief description of the Charm++ environment and of the parallel version of AMIGO algorithm. Finally, in Section 4 experimental results and evaluations of our parallel implementation on a distributed system are shown.

2 Parallel Issues in Interval B&B Global Optimization

The verified global optimization method considered in this paper belongs to the B&B class of methods, where the given initial problem is successively subdivided into smaller subproblems. Some of these subproblems are again subdivided while other do not need to be considered anymore because it is known they cannot contain the solution of the problem. Parallelizing B&B methods mainly consists of distributing among processors the set of independent subproblems being dynamically created. In order to achieve an efficient parallel method one should be concerned with the following issues:

1. All processors should always be busy handling subproblems;
2. The total cost of handling all the subproblems should not be greater than the cost of the serial method;
3. The overhead due to communications among processors should be small.

More precisely, issue 2 means that all processors should not be just busy but doing useful work. It is necessary to point out that a B&B algorithm executed in parallel does not process the subproblems in the same order that a sequential program does, so the number of created (and eliminated) subproblems will depend on the number of processors used in a particular execution. The resulting effect is that a specific parallel execution will create more (or sometime less) subproblems depending on the specific function and the number of processors.

Here we shall investigate the parallelization of a B&B global optimization algorithm (AMIGO) on a distributed memory multicomputers. In this case it is difficult to fulfill all the above described issues (1-3), simultaneously. For issue 2, it is important that the current best bounding criterion (in our case the smallest value of $\tilde{f}$ on all processors) is sent to every processor as soon as possible. Additionally, one must try to distribute among processors all the subproblems created thus far. This will contribute to keep all processors equally loaded or at least to keep them all busy. In addition one should try to fulfill all three issues to get an efficient parallel method.

When parallelizing the B&B method there are two possibilities for managing subproblems. The first is to store subproblems on one central processor. The other is to distribute them among processors. In our context of verified global optimization this means either to store boxes in a list on one processor or to store them in several lists each created on every processor. The first case has a disadvantage: the maximal length of a list is limited by the amount of memory of one processor, whereas in the second case the memory of all processors can be used.

In [12], where parallelization of different methods for verified global optimization was investigated, one can find a very simple strategy where boxes are stored in a central list which can be handled by all processors. Similar parallelizations of this master-slave principle were proposed in [5] and [1]. Contrarily, Eriksson manages processors in a ring where each processor has its own list of not processed subproblems [4].

3 Parallel Implementation in Charm++

The main characteristics of Charm++, an object oriented portable parallel programming language based on C++, are introduced here to describe our parallel algorithm.

What sets Charm++ apart from traditional programming models such as message passing and shared variable programming is that the execution model of Charm++ is message-driven. Therefore, computations in Charm++ are triggered based on arrival of associated messages. These computations in turn can fire off more messages to other (possibly remote) processors that trigger more computations on those processors. Some of the programmer-visible entities in a Charm++ program are:

Concurrent Objects (Chares): A chare is a Charm++ object that can be created on any available processor and can be accessed from remote processors. A chare is similar to a process. Chares are created dynamically, and many chares may be active simultaneously.

Communication Objects (Messages): Chares send messages to one another to invoke methods asynchronously. Conceptually, the system maintains a "work-pool" consisting of seeds for new chares, and messages for existing chares.

Every Charm++ program must have at least one mainchare. Charm++ starts a program creating a single instance of each mainchare on processor 0, and invokes constructor methods of these chares. Typically, these chares then create a number of other chares, possibly on other processors, which can simultaneously work to solve the problem at hand.

Each chare contains a number of entry methods, which are methods that can be invoked from remote processors. In Charm++, the only communication method among processors is an invocation to an asynchronous entry method on remote chares. For this purpose, Charm Kernel needs to know the types of chares in the user program, the methods that can be invoked on these chares from remote processors, the arguments these methods take as input, etc.

Charm++ also permits prioritizing executions (associating priorities with method invocations), conditional message packing and unpacking (for reducing messaging overhead), quiescence detection (for detecting completion of some phase of the program), and dynamic load balancing (during remote object creation).

In our parallel program, a chare has two entry methods:

Process-Box: It execute one iteration of AMIGO algorithm over the box received in the given message.
Update-$\tilde{f}$: It upgrade the current $\tilde{f}$ value in the current chare.

We use a static mapping of chares on processors and there is only one chare in one processor. Therefore, we do not use the Charm++ dynamic load balancing capability. Each chare will be triggered when receives a message which invokes its entry methods. A chare also can generate messages for other chares (or itself). For instance, when a message with a Process-Box entry method arrives to a chare in one processor, the entry method can:

– Reject the box by some elimination rule: No messages are generated.
– Divide the box: Two new sub-boxes are generated and two new messages with a Process-Box entry method are generated. The receiver (chare) of these new messages are randomly selected.
– A solution box was found: A message with the solution box is generated and sent to the mainchare.

The possibility of associating priorities to entry method invocations is very important in our model. For instance if one chare obtains a better value of $\tilde{f}$, this value has to be broad-casted to all the chares, then they will apply the Cutoff test as soon as possible. Therefore, the Update-$\tilde{f}$ messages will have more priority that the Process-Box ones. If a Process-Box message arrives and its associated box satisfies the Cutoff test then it will not be processed. A priority also has been established in the Process-Box messages to first process the more promising boxes, i.e, those which let to obtain a better $\tilde{f}$ value. This tries to minimize the possibility that the search region visited by the parallel version will be larger than the visited by the sequential one.

4 Performance Results

The speed-up and work load imbalance for our parallel implementation of AMIGO algorithm are shown in Figures 1 and 2, respectively. All the data where obtained from executions carried out on a cluster of workstations composed of 16 nodes with two Pentium XEON 3Ghz, with Hyper-threading running Linux. The nodes are connected by a Gigabit Ethernet network.

Table 1 shows the set of test problems used to evaluate the algorithms. T_1 correspond with the execution time in seconds of AMIGO algorithm and ε_1 with the precision reached by AMIGO to obtain a solution in less than one hour of running time [7]. T_2 and ε_2 are analogous but for the parallel algorithm running in one processor. Our experiments were done in such a way that executions spend less that one hour. It means that increasing the accuracy of ε_1 and ε_2 one order of magnitude, algorithms do not provide any solution after running one hour. From results in Table 1 is clear that the parallel version can reach better precision. A possible reason is that the parallel version do not need to handle the storage of the pending boxes, just to process the entry methods.

Figure 1 clearly shows that, for this set of very hard to solve problems, an average linear speed-up was obtained.

Table 1. Comparison between sequential AMIGO (T_1, ε_1) and AMIGO-Charm++ (T_2, ε_2) algorithms

Name	Ref	n	$T_1(sec.)$	ε_1	$T_2(sec.)$	ε_2
Schwefel 2.14 (Powell)	[11]	4	15,85	10^{-5}	1898,97	10^{-7}
Schwefel 3.1p	[11]	3	1302,67	10^{-4}	24,14	10^{-4}
Ratz 5	[11]	3	667,35	10^{-3}	2245,77	10^{-5}
Ratz 6	[11]	5	823,49	10^{-3}	1007,58	10^{-4}
Ratz 7	[11]	7	903,75	10^{-3}	2459,16	10^{-4}
Schwefel 2.10 (Kowalik)	[14]	4	405,54	10^{-2}	3116,71	10^{-9}
Griewank 10	[13]	10	334,04	10^{-2}	1114,49	10^{-14}
Rosenbrock 10	[3]	10	199,19	10^{-2}	1357,70	10^{-14}
Neumaier 2	[9]	4	84,41	10^{-2}	1615,26	10^{-14}
EX2	[2]	5	44,38	10^{-2}	3302,40	10^{-10}
Ratz 8	[11]	9	10,65	10^{-2}	685,48	10^{-3}

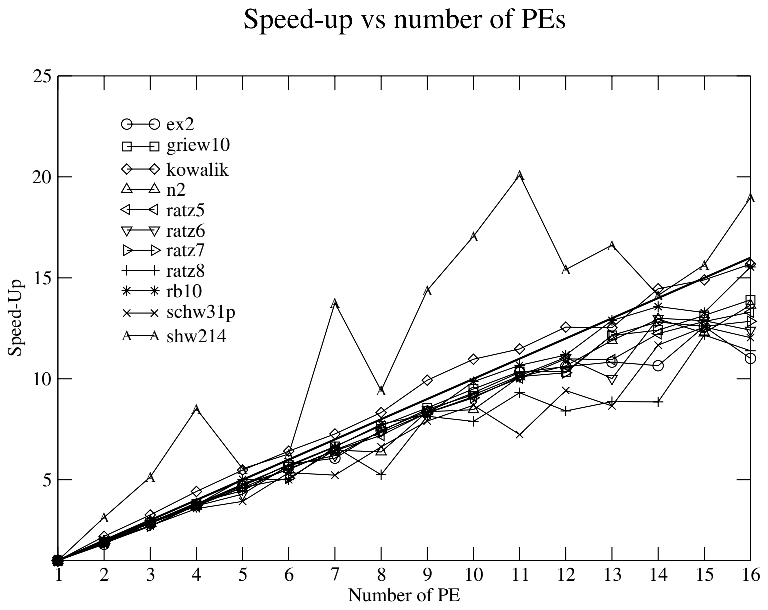


Fig. 1. Speed-up

The workload imbalance has been obtained as $(L_{max} - L_{av})/L_{av} \in [0, p-1]$; where L_{max} is the maximum workload and L_{av} is the average workload in a set of p processors. Notice (see Figure 2) that the workload imbalance in all the cases is negligible.

As the numerical results show, the parallel implementation of AMIGO using Charm++ is suitable to obtain solutions with near linear and sometimes with super speed-ups.

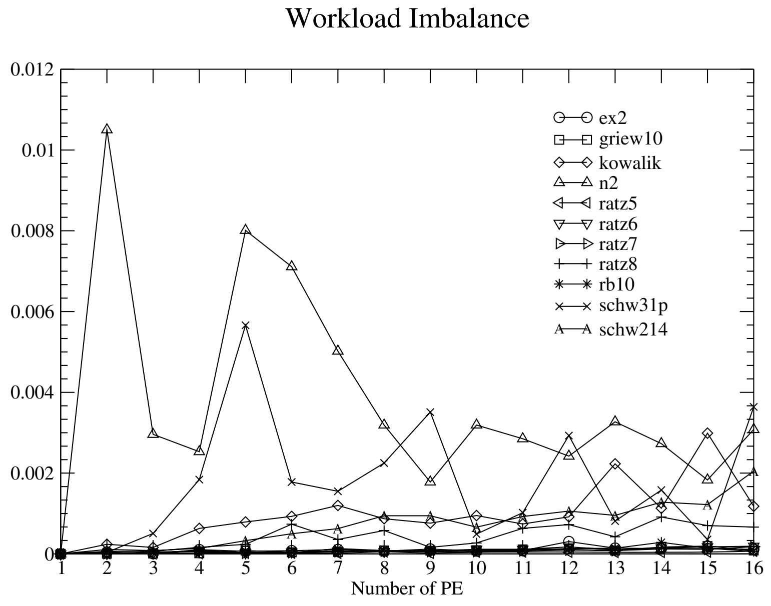


Fig. 2. Workload imbalance

Acknowledgement

This work was supported by the Ministry of Education of Spain(CICYT TIC2002-00228).

References

1. Berner, S. Parallel methods for verified global optimization, practice and theory. *Journal of Global Optimization* 9:1–22, 1996.
2. Csendes, T. and D. Ratz: 1997, 'Subdivision Direction Selection in Interval Methods for Global Optimization'. *SIAM Journal of Numerical Analysis* **34**, 922–938.
3. Dixon, L.W.C. and G.P. Szego (eds.): 1975, *Towards Global Optimization*. North-Holland Publishing Company.
4. Eriksson, J., Lindström, P.: A parallel interval method implementation for global optimization using dynamic load balancing. *Reliable Computing* 1:77-91, 1995.
5. Henriksen, T., Madsen, K. Use of a depth-first strategy in parallel Global Optimization. *Technical Report 92-10*, Institute for Numerical Analysis, Technical University of Denmark, 1992.
6. Kearfott, R.B. Rigorous Global Search: Continuous Problems. Kluwer Academic Publishers, Dordrecht, Holland, 1996.
7. Martínez, J.A., Casado, L.G., García, I., Tóth, B.: AMIGO: Advanced Multidimensional Interval Analysis Global Optimization Algorithm. In Floudas, C., Pardalos, P., eds.: *Nonconvex Optimization and Applications Series. Frontiers in Global Optimization.* 74:313-326. Kluwer Academic Publishers, 2004.

8. Moore, R.: Interval analysis. Prentice-Hall, 1966.
9. Neumaier, A.: Interval Methods for Systems of Equations. Cambridge University Press, 1990.
10. Ratschek, H., Rokne, J.: New Computer Methods for Global Optimization. Ellis Horwood Ltd., 1988.
11. Ratz, D., Csendes, T.: On the selection of subdivision directions in interval branch and bound methods for global optimization. *Journal of Global Optimization* 7:183-207, 1995.
12. Suiunbek, I.: A New Parallel Method for Verified Global Optimization. *PhD thesis*, University of Wuppertal, 2002.
13. Törn, A. and A. Žilinskas: 1989, *Global Optimization*, Vol. 3350. Berlin, Germany: Springer-Verlag.
14. Walster, G., E. Hansen, and S. Sengupta: 1985, 'Test results for global optimization algorithm'. *SIAM Numerical Optimization 1984* pp. 272–287.

On the Approximation of Interval Functions

Klaus Meer*

Department of Mathematics and Computer Science
Syddansk Universitet, Campusvej 55, 5230 Odense M, Denmark
meer@imada.sdu.dk

Abstract. Many problems in interval arithmetic in a natural way lead to a quantifier elimination problem over the reals. By studying closer the precise form of the latter we show that in some situations it is possible to obtain a refined complexity analysis of the problem. This is done by structural considerations of the special form of the quantifiers and its implications for the analysis in a real number model of computation. Both can then be used to obtain as well new results in the Turing model. We exemplify our approach by dealing with different versions of the approximation problem for interval functions.

1 Introduction

When studying a problem in interval arithmetics one of the basic assumptions is that the input data is not known accurately. Instead of dealing with say an input sequence $(x_1, \dots, x_n)$ of rationals describing our precise data, we consider a sequence of intervals $[\underline{x}_1, \overline{x}_1], \dots, [\underline{x}_n, \overline{x}_n]$ such that the actual data points x_i are only known to belong to $[\underline{x}_i, \overline{x}_i], 1 \le i \le n$.

Example 1. (see [11]) Consider the problem: Given a matrix $A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^m$, is there a vector $x \in \mathbb{R}^n$ such that $Ax = b$? As a problem in interval arithmetic we should start with an *interval matrix* $\mathfrak{A}$, i.e. a set of intervals $[\underline{a}_{ij}, \overline{a}_{ij}] \subseteq \mathbb{R}$ for each position (i, j) and an *interval vector* $\mathfrak{b}$ of intervals $[\underline{b}_i, \overline{b}_i]$ as components. The only knowledge about the original data (A, b) then is the information $A \in \mathfrak{A}$, i.e. $\forall i, j \; a_{ij} \in [\underline{a}_{ij}, \overline{a}_{ij}]$ and similarly for $b \in \mathfrak{b}$. A typical question then could be: Is there any choice $A \in \mathfrak{A}, b \in \mathfrak{b}$ such that there is an $x \in \mathbb{R}^n$ with $Ax = b$?

The above interval linear systems problem is known to be NP-hard [8]. From an informal point of view it is easy to get an intuition why this is the case. The interval framework "automatically" introduces a linear number (in the problem dimension) of quantifiers ranging over the *real* numbers; they are used to describe the interval information available about the input data:

$$\exists\, A \in \mathfrak{A} \;\exists\, b \in \mathfrak{b} \;\exists\, x \in \mathbb{R}^n \; Ax = b \; .$$

* Partially supported by the EU Network of Excellence PASCAL Pattern Analysis, Statistical Modelling and Computational Learning and by the Danish Natural Science Research Council SNF. This publication only reflects the author's views.

The first two blocks of existential quantifiers are not present in the exact setting and increase the problem's complexity. They naturally lead to the consideration of *algebraic* models of computation like the BSS model introduced by Blum, Shub, and Smale, [2].

In real-life computers, operations with real numbers are hardware supported. Almost universally, floating-point arithmetic is used for scientific computation. Therefore, if we have a certain number of bit operations as part of a single operation with real numbers, then these operations will be performed much faster than if they were unrelated operations with bits. From this viewpoint, in addition to the standard Turing complexity, it is desirable to investigate the complexity in terms of how many operations with real numbers are necessary. This cost measure is studied in the real number BSS model of computation (resp. in algebraic complexity theory). The approach is substantiated by the fact that (see [17]) "most practioneers of scientific computing do not experience much difference between the real number model and floating point arithmetic." Of course, there are situations in which a more careful analysis has to be performed. However, for numerically stable algorithms the real number model seems to reflect pretty well the floating-point costs. Moreover, as already mentioned, in the framework of interval arithmetic the real number model comes in quite naturally already through the formulation of problems.

Note as well that in the BSS model the problem of deciding general existential formulas in the first-order theory $\mathrm{FO}_{\mathbb{R}}$ over the reals is $\mathrm{NP}_{\mathbb{R}}$-complete, where $\mathrm{NP}_{\mathbb{R}}$ denotes a real analogue of NP. For more details see [2].

Example 2. This example deals with the best linear approximation of quadratic interval functions (BLA for short, see [10,8]). A problem instance is given by a box $B = [\underline{b}_1, \overline{b}_1] \times \ldots \times [\underline{b}_n, \overline{b}_n] \subseteq \mathbb{R}^n$, a bound $M \in \mathbb{R}^n$ and a quadratic interval function $f : B \mapsto \{\text{intervals in } \mathbb{R}\}$. The latter means that there are two quadratic polynomials $\underline{f}, \overline{f} : B \mapsto \mathbb{R}$ such that $\forall\, y \in B\ \underline{f}(y) \leq \overline{f}(y)$ and the only information we have about f is that $\forall\, y \in B\ f(y) \in [\underline{f}(y), \overline{f}(y)]$. The task is to approximate f as best as possible with respect to the maximum norm on B by two linear functions $\underline{X}, \overline{X} : B \to \mathbb{R}$, i.e. to compute

$$\min_{\underline{X}, \overline{X}} \max_{y \in B} \overline{X}(y) - \underline{X}(y) \leq M$$

under the constraints

$$\forall\, y \in B : \quad \begin{array}{l} \overline{X}(y) \geq \overline{f}(y) \\ \underline{X}(y) \leq \underline{f}(y) \end{array}$$

As decision problem we ask whether the minimal value is at most M. This problem is a linear semi-infinite optimization problem that was studied in [10]. There, it was shown that the problem (when restricted to rational input data) is NP-hard. However, no upper bound is known, i.e. the problem is not known to belong to a certain level of the polynomial hierarchy. It is neither known what the complexity for a fixed input dimension n is (Koshelev et al. [10] showed polynomial solvability for $n = 1$.)

For our approach it is most interesting to see that the BLA problem logically can be expressed as a $\Sigma_2^{\mathbb{R}}$ formula in the first order theory over the reals. The first block of existential quantifiers asks for the linear functions $\underline{X}, \overline{X}$, the second block of universal quantifiers checks the constraints. The entire formula thus has the shape

$$\exists\, \underline{X}, \overline{X} \in \mathbb{R}^n \;\forall\, y \in B \;\Phi(\underline{X}, \overline{X}, y) \;,$$

where Φ is a quantifier free $\mathrm{FO}_{\mathbb{R}}$ formula. However, note that this does not automatically imply the problem to belong to Σ_2 in the classical polynomial hierarchy when inputs are restricted to be rational numbers.

In this extended abstract we show how a further analysis of the logical structure of the above formula results in refined and even new complexity results concerning the BLA problem, both in the Turing and the BSS model. For full details on this part see [13]. We then also discuss more generally the limits of our approach.

2 Framework and Techniques

We shall use Example 2 to clarify our general ideas. Given the real quantifiers naturally involved in many such interval problems we start from real number complexity theory as described in [2]. In this framework we perform an analysis that allows us as well to obtain new and to refine known results for the problem's complexity in the Turing model. In particular, we show:

- many of the NP-hard interval problems do likely not capture the full complexity of the analogue class $\mathrm{NP}_{\mathbb{R}}$;
- for expressing many such problems the full power of real quantifier elimination is not necessary;
- a new upper bound for the BLA problem in the Turing model: BLA $\in \Sigma_2$;
- a new fixed parameter result: BLA is polynomial time solvable in the Turing model for fixed variable dimension $n \in \mathbb{N}$.

The proofs of the above results are divided into two main parts. In the first we analyse consequences the logical description of a problem has with respect to its real number complexity. In the second we check the particular problem we are interested in with respect to the special logical form it can be expressed in. Whereas the first part is using structural complexity theory in the BSS model, the second part is depending very much on the problem under consideration. In case of the BLA problem, the Reduction Ansatz from semi-infinite optimization together with arguments from [12] are crucial.

2.1 Digital Quantifiers

Similar to the class NP in the BSS model the real analogue $\mathrm{NP}_{\mathbb{R}}$ can be characterized as all real number decision problems A such that there is a (real) decision problem $B \in \mathrm{P}_{\mathbb{R}}$ together with a polynomial p such that

$$A = \{x \in \mathbb{R}^n | \exists\, y \in \mathbb{R}^m \; m \le p(n) \text{ and } (x, y) \in B \text{ for some } n, m\} \;.$$

The class $\mathrm{DNP}_{\mathbb{R}}$ is the subclass of problems for which in the above characterization it is sufficient to let the existential quantifiers range over some $\{0,1\}^m$, only, see [3].

Thus, a problem is in $\mathrm{DNP}_{\mathbb{R}}$ if the verification procedure can be performed using a proof from a finite search space instead of one from an uncountable set $\mathbb{R}^m$. Clearly, it is $\mathrm{P}_{\mathbb{R}} \subseteq \mathrm{DNP}_{\mathbb{R}} \subseteq \mathrm{NP}_{\mathbb{R}}$. Similarly for the real version $\Sigma_2^{\mathbb{R}}$ of Σ_2 (and for all the other levels of the real polynomial time hierarchy) we can define the digital subclass $\mathrm{D}\Sigma_2^{\mathbb{R}}$. Though we cannot expect to be able to prove $\mathrm{DNP}_{\mathbb{R}} \neq \mathrm{NP}_{\mathbb{R}}$ or $\mathrm{D}\Sigma_2^{\mathbb{R}} \neq \Sigma_2^{\mathbb{R}}$ it is possible to substantiate these conjectures by the theorem below. Before stating it let us shortly explain two notions used in the theorem.

Weak reductions in the BSS model were introduced by Koiran [7] as a way to penalize computations that produce high degree polynomials by repeated squaring. For example, under the weak cost measure performing n times a repeated squaring of an input $x \in \mathbb{R}$ has exponential cost since the polynomial generated when x is seen as a variable has degree 2^n. For a real number complexity class $\mathcal{C}$ we denote by the superscript $\mathcal{C}^w$ the corresponding real number complexity classes when implicit complexity statements refer to the weak cost measure. For example, $\mathrm{DNP}_{\mathbb{R}}^w$ is the class of problems verifiable in weak polynomial time by guessing a sequence in $\{0,1\}^\infty$. Similarly, $\mathrm{D}\Sigma_2^{\mathbb{R},w}$ is the weak version of $\mathrm{D}\Sigma_2^{\mathbb{R}}$.

The family of (particular) resultant polynomials RES_n is known from algebraic geometry, see f.e. [5] and [16]:

Definition 1. *Let* $n \in \mathbb{N}, N := \frac{1}{2} \cdot n^2 \cdot (n+1)$. *The* resultant polynomial $RES_n : \mathbb{R}^N \to \mathbb{R}$ *is a polynomial which as its indeterminates takes the coefficient vectors of* n *homogeneous degree two polynomials. It is the unique (up to sign) irreducible polynomial with integer coefficients that generates the variety of (coefficient vectors of) systems that have a non-trivial complex zero. The set of these systems is denoted by* $\mathcal{H}$*, the solvable ones by* $\mathcal{H}_0$.

Resultants are conjectured not to be efficiently computable, see [14] for some hardness results and [5] for the general theory of resultants.

Theorem 1. *a) No problem in* $\mathrm{DNP}_{\mathbb{R}}^w$ *is* $\mathrm{NP}_{\mathbb{R}}$*-complete under weak polynomial time reductions. No problem in* $\mathrm{D}\Sigma_2^{\mathbb{R},w}$ *is* $\mathrm{NP}_{\mathbb{R}}$*-hard under weak polynomial time reductions.*
b) Suppose there is no *(non-uniform) polynomial time algorithm which for each* $n \in \mathbb{N}$ *computes a non-zero multiple of* RES_n *on a Zariski-dense subset of* $\mathcal{H}_0$*. Then* no *problem in* $\mathrm{DNP}_{\mathbb{R}}$ *is* $\mathrm{NP}_{\mathbb{R}}$*-complete and* <u>*no*</u> *problem in* $\mathrm{D}\Sigma_2^{\mathbb{R}}$ *is* $\mathrm{NP}_{\mathbb{R}}$*-hard under polynomial time reductions in the BSS model.*
c) Part b) also holds with respect to Turing instead of many-one reductions. It is as well true for computational (instead of decision) problems that can be solved in polynomial (real) time using oracle calls to a $\mathrm{DNP}_{\mathbb{R}}$*-oracle.*

Proof. This is done by generalizing related results in [4] and [12]. Part a) follows almost straightforwardly from those results. For part b) assume the hardness of the problems under consideration. Now study the following real number decision problem: Given a system f of n homogeneous real polynomials of degree 2, is there a common non-trivial (complex) zero. This problem clearly belongs to $\mathrm{NP}_{\mathbb{R}}$ and thus, under the assumption that the theorem is false, can be polynomially reduced to a $\mathrm{D}\Sigma_2^{\mathbb{R}}$ problem (respectively one in $\mathrm{DNP}_{\mathbb{R}}$). This potential reduction is now combined with the Dubois-Risler real Nullstellensatz. It is then shown that the Nullstellensatz implies the claimed efficient computation of a non-zero multiple of RES_n to be part of the reduction algorithm. Part c) follows directly from the proof of part b): Also from a potential Turing reduction the desired algorithm for computing the resultant polynomials (modulo the cited conditions) can be designed. □

Since the computation of RES_n is conjectured to be difficult, the above theorem informally can be phrased as: A problem in $\text{DNP}_\mathbb{R}$ or in $\text{D}\Sigma_2^\mathbb{R}$ is $\text{NP}_\mathbb{R}$-hard only if the potentially difficult problem of computing RES_n is efficiently solvable (modulo the above conditions). We thus take a proof that a $\Sigma_2^\mathbb{R}$-problem actually belongs to $\text{D}\Sigma_2^\mathbb{R}$ as indication for it not to be $\text{NP}_\mathbb{R}$-hard. Concerning weak classes and reductions the statements are absolute; note that for the weak versions of the real number complexity classes $\text{P}_\mathbb{R}$ and $\text{NP}_\mathbb{R}$ their inequality was proven in [4].

2.2 BLA Lies in $\text{D}\Sigma_2^\mathbb{R}$

For a concrete problem in $\Sigma_2^\mathbb{R}$ it might of course be unclear whether it belongs to $\text{D}\Sigma_2^\mathbb{R}$ and how to prove it. In case of the BLA problem this can be achieved by combining techniques from semi-infinite optimization and structural results from classical quadratic programming. The proof will not only establish $\text{BLA} \in \text{D}\Sigma_2^\mathbb{R}$ but allows as well to conclude new results for BLA in the Turing model.

Theorem 2. *a) The* BLA *problem with real input data lies in*

$$D\Sigma_2^\mathbb{R}\ (actually\ in\ D\Sigma_2^{\mathbb{R},w}).$$

b) The BLA *problem with rational input data lies in* Σ_2.
c) For fixed variable number $n \geq 1$ both in the BSS and in the Turing model BLA *is solvable in polynomial time.*

Proof. The main work of the proof is to show part a). The overall idea is to find a solution $(\underline{X}, \overline{X})$ and verify its correctness as linear approximation as well as verify whether this approximation realizes the demanded bound M of the maximum norm of $\overline{X} - \underline{X}$ on B. This program is performed along the following steps

1) Assuming $(\underline{X}, \overline{X})$ to be known we guess certain discrete information which then is used to compute at least two points $y^{(i)}$ and $y^{(j)}$ satisfying necessary optimality conditions resulting from the Reduction Ansatz of semi-infinite optimization, see [6]. This reduces the infinitely many $y \in B$ to finitely many. However, these finitely many points still are real, i.e. this step does not yet remove the real quantifiers.
2) From step 1) we deduce conditions that have to be fulfilled by an optimal solution $(\underline{X}, \overline{X})$. These conditions lead to a linear programming problem. The solution of the latter, using additional work, can be shown to be computable by a digital $\Sigma_1^\mathbb{R}$ algorithm.
3) Finally, the candidate obtained in 2) is checked for optimality. This mainly requires to check the constraints, which result in two quadratic programs in y representing the lower level of the initial semi-infinite problem. Using the results of [12] this problem belongs to class *co*-$\text{DNP}_\mathbb{R} = D\Pi_\mathbb{R}^1$. Together, we obtain a $\text{D}\Sigma_2^\mathbb{R}$ algorithm.

We remark that the above analysis actually implies all computations to be executable in the corresponding weak classes as well.
b) The precise analysis of the proof of part a) shows that in case of rational input data the numbers of 0's and 1's quantified in the verification procedure are only chosen to select certain positions in the initial matrices and vectors giving the problem coefficients. They do neither depend on intermediately generated rationals of larger bit-size nor on newly introduced real constants. It follows membership in Σ_2 for rational inputs.
c) Similarly, the proof of a) shows that the number of quantified variables in the corresponding Σ_2 formula is a (linear) function of n. Thus, if n is fixed we can eliminate the quantifiers in polynomial time by evaluating all possible 0-1 assignments for the bound variables. Finally, a property in complexity class P has to be checked. □

Part a) above allows to conclude that BLA is unlikely to be $\mathrm{NP}_{\mathbb{R}}$-hard under full and is not $\mathrm{NP}_{\mathbb{R}}$-hard under weak polynomial reductions. Part b) gives a new upper bound for the rational version extending the NP-hardness result in [10]. Finally, part c) answers an open question of [8].

3 How Far Does the Approach Lead?

A huge number of NP-hard problems in interval arithmetics is studied in [8]. One might ask in how far the above methods will work at least for some of those problems as well. It should be clear that there are a lot of interval problems not only NP-hard in the Turing model, but also in the real number framework. If we start from an algebraic problem that already in its exact form is $\mathrm{NP}_{\mathbb{R}}$-hard (when real inputs are considered), then its interval version maintains this property as well.

Let us clarify this by some examples. The problems of deciding the existence of a common real root for a system of quadratic (real) polynomials or existence of a real root of a single (real) degree 4 polynomial are well known to be $\mathrm{NP}_{\mathbb{R}}$-complete [2]. Thus, as soon as one of these problems is involved in the exact formulation the interval version is at least as hard and the above techniques do not apply. An extremely important question is where the boundary is. For the Best Linear Approximation problem of Example 2 the decisive properties are

- the Reduction Ansatz and
- the structural properties of (non-convex) quadratic programming allowing for a discrete coding of optimal points.

It is unclear whether, in particular, the techniques related to the second property can be extended to higher degree polynomials as objective functions. The above result is based on the fact that quadratic programs (i.e. a quadratic polynomial as objective function together with linear constraints) attain their infimum if they are bounded from below. This results in the possibility to reduce the problem to linear programs by using necessary first order optimality conditions. And the Linear Programming decision problem belongs to class $\mathrm{DNP}_{\mathbb{R}}$.

If we consider degree 4 polynomials as objective functions everything is different. The related approximation problem *is* $\mathrm{NP}_{\mathbb{R}}$-hard.

Before we prove this result we state the

Theorem 3. *Given $n \in \mathbb{N}$, a compact box $B \subset \mathbb{R}^n$ and a degree 4 polynomial $f \in \mathbb{R}[x_1, \ldots, x_n]$ that satisfies $f(x) \geq 0$ for all $x \in B$, it is NP$_{\mathbb{R}}$-complete to decide whether f has a real zero in B. We denote this problem by BOX-4-FEAS.* □

Due to space limitations we postpone the proof of this result to a future paper [9]. It is basically a straightforward though tedious application of the known quantifier elimination algorithms and their complexity bounds as given, for example, in [1].

Theorem 4. *Consider the problem of finding the best linear approximation of an interval function of degree 4 (cf. Example 2): Given a (compact) box $B \subset \mathbb{R}^n$, a quartic interval function $f : B \mapsto \{$intervals in $\mathbb{R}\}$, i.e. there are two degree 4 polynomials $\underline{f}, \overline{f} : B \mapsto \mathbb{R}$ such that $\forall\, y \in B\ \underline{f}(y) \leq f(y) \leq \overline{f}(y)$. Find two linear functions $\underline{X}, \overline{X} : B \to \mathbb{R}$ that solve the constrained optimization problem*

$$\min_{\underline{X}, \overline{X}} \max_{y \in B} \overline{X}(y) - \underline{X}(y)$$

under the constraints

$$\forall\, y \in B : \quad \begin{aligned} \overline{X}(y) &\geq \overline{f}(y) \\ \underline{X}(y) &\leq \underline{f}(y). \end{aligned}$$

Then this problem is NP$_{\mathbb{R}}$-hard.

Similarly, for the decision version it holds: Given, in addition to f, n and B, a bound $M \in \mathbb{R}$, does the best linear approximation $(\overline{X}, \underline{X})$ satisfy

$$\max_{x \in B} |\overline{X}(x) - \underline{X}(x)| \leq M\,?$$

is NP$_{\mathbb{R}}$-complete.

Proof. We reduce the BOX-4-FEAS problem to the decision problem under consideration. Given n, f and B as input for the former it is easy to compute an upper bound $M \in \mathbb{R}$ such that $f(x) \leq M$ for all $x \in B$. Just plug in the largest possible value B for all x_i and sum up the absolute values of all monomials occuring in f. This bound clearly is computable in polynomial time.

As input for the approximation problem we now choose

$$\overline{f}(x) := M\ \forall\, x \in B\ ,\ \underline{f}(x) := f(x)\ \forall\, x \in B\ .$$

Since $\overline{f}$ is a constant function it follows that the optimal choice for $\overline{X}$ is as well the constant function M. As continuous function $f = \underline{f}$ attains its infimum on the compact box B, say in a point $x^* : f(x^*) = \min_{x \in B} f(x)$. By assumption, $f(x) \geq 0$ for all choices of x. Let $\underline{X}$ be an optimal choice (there might be several) for the approximation problem. It is clear that

$$\max_{x \in B} \overline{X}(x) - \underline{X}(x) = \max_{x \in B} M - \underline{X}(x) \leq M,$$

since $\underline{X}(x) := 0$ is a suitable candidate for computing $\min_{\underline{X}} \max_{x \in B} M - \underline{X}(x)$. In the latter inequality we actually get equality if and only if f has a real zero in B. Otherwise, the choice $\underline{X}(x) := f(x^*) > 0$ gives a better (the optimal!) result. □

The theorem shows that if we share the general believe that DNP$_{\mathbb{R}} \neq$ NP$_{\mathbb{R}}$, then a limit for our techniques is reached with the linear approximation problem for interval functions of degree 4.

Open Problem: What's about the real number complexity of the best linear approximation problem for (polynomial) interval functions of degree 3? In particular: Does the decision version belong to DNP$_{\mathbb{R}}$?

The above arguments are closely related to the range problem of interval functions. This issue will be discussed in more detail in [9].

For many other problems related to *linear* interval systems (for a collection of such problem see Chapters 11 and 12 in [8]) it seems that our results can be applied. They yield some more optimistic upper bounds than what could be feared from solving them with general quantifier elimination methods. It might be fruitful to analyze further such problems under this point of view.

We give one such result here. It deals with the computation of a solution interval of an interval linear system. As in Example 1 let $\mathfrak{A} := [\underline{A}, \overline{A}]$ be an interval matrix and $\mathfrak{b} := [\underline{b}, \overline{b}]$ an interval vector in $\mathbb{R}^m$.

Definition 2. *(see [8]) a) A* possible solution *of the interval linear system $\mathfrak{A} \cdot x = \mathfrak{b}$ is a vector $y \in \mathbb{R}^n$ such that there exist $A \in \mathfrak{A}, b \in \mathfrak{b}$ for which $A \cdot y = b$.*
b) By the j-th solution interval $\mathbf{y}_j = [\underline{y}_j, \overline{y}_j]$ *of the system we mean a (possibly infinite) interval such that for all possible solutions y of the system the j-th component y_j of y belongs to $[\underline{y}_j, \overline{y}_j]$. Moreover, the values $\underline{y}_j, \overline{y}_j$ do occur as components of certain possible solutions (i.e. they are the minimal and maximal such components).*

We consider the problem of computing the solution intervals of an interval linear system. It was shown by Rohn [15] that

Theorem 5. *(Rohn) The problem of computing the $\mathbf{y}_j$'s (or even computing what is called a* possibly finite enclosure*) is NP-hard in the Turing model.*

Here, we show that in the real number framework Theorem 2, part c) applies, i.e. the problem likely does not share the full difficulty of class NP$_{\mathbb{R}}$.

Theorem 6. *The problem to compute the solution intervals of an interval linear system can be solved in $FP_{\mathbb{R}}^{DNP_{\mathbb{R}}}$, i.e. by a polynomial time real number oracle algorithm that has access to an oracle for problems in DNP$_{\mathbb{R}}$. Thus, Theorem 2, part c) applies.*

Proof. Let $\mathfrak{A}, \mathfrak{b}$ be the data of an interval linear system. We use the following result from [11] to characterize the set $\mathcal{R}$ of possible solutions of $\mathfrak{A} \cdot x = \mathfrak{b}$:

$$\mathcal{R} = \{y^{(1)} - y^{(2)} | y^{(1)} \geq 0,\ y^{(2)} \geq 0,\ {y^{(1)}}^T \cdot y^{(2)} = 0,$$
$$\underline{A} \cdot y^{(1)} - \overline{A} \cdot y^{(2)} \leq \overline{b},\ \overline{A} \cdot y^{(1)} - \underline{A} \cdot y^{(2)} \geq \underline{b}\}.$$

Clearly, for $1 \leq j \leq n$ the endpoints $\underline{y}_j, \overline{y}_j$ of the solution interval $\mathbf{y}_j$ are given as solutions of the following optimization problems: $\underline{y}_j$ is the solution of

$$\begin{aligned}&\min_{y^{(1)},y^{(2)} \in \mathbb{R}^n} \; y_j^{(1)} - y_j^{(2)} \text{ subject to the constraints}\\ &y^{(1)} \geq 0,\; y^{(2)} \geq 0,\; {y^{(1)}}^T \cdot y^{(2)} = 0,\\ &\underline{A} \cdot y^{(1)} - \overline{A} \cdot y^{(2)} \leq \overline{b},\; \overline{A} \cdot y^{(1)} - \underline{A} \cdot y^{(2)} \geq \underline{b}\,.\end{aligned}$$

The solution of the corresponding maximization problem is $\overline{y}_j$.

Given the previous results we now can easily design an $\mathrm{FP}_{\mathbb{R}}$ algorithm for computing the solutions of these optimization problems. First, we use the $\mathrm{DNP}_{\mathbb{R}}$ oracle to remove the complementary slackness condition ${y^{(1)}}^T \cdot y^{(2)} = 0$ by figuring out which components of the two vectors are zero in an optimal solution. This clearly is a $\mathrm{DNP}_{\mathbb{R}}$ problem since we only have to code the component indices. We plug in the corresponding zero values into the problem. This turns it into a Linear Programming problem. As mentioned before, a solution of a Linear Programming problem (with real data) can be computed using a $\mathrm{DNP}_{\mathbb{R}}$ oracle, thus finishing the proof. □

Let us finally mention that, once more using the characterizations given in [11], similar results hold as well for the linear interval systems problem from Example 1 and some related variants, see [13].

4 Conclusions

We have seen that some problems in interval arithmetic in a very natural manner give rise to study the problem in the BSS model. A concise analysis concerning the logical expressibility of a problem then allows to obtain structural complexity results not only in the BSS model but also in the Turing model. The latter can refine investigations that do not go further than stating NP-hardness of a problem as soon as real quantifiers come into play.

Acknowledgement

I want to thank G.W. Weber for helpful discussions on semi-infinite programming. Thanks are due to V. Kreinovich and two anonymous referees for useful comments concerning the preparation of this paper.

References

1. S. Basu, R. Pollack, M.F. Roy. On the combinatorial and algebraic complexity of quantifier elimination. *Journal of the ACM*, 43(6), 1002–1045, 1996.
2. L. Blum, F. Cucker, M. Shub, S. Smale. Complexity and Real Computation. Springer, 1998.
3. F. Cucker, M. Matamala. On digital nondeterminism. *Mathematical Systems Theory*, 29, 635–647, 1996.

4. F. Cucker, M. Shub, S. Smale. Complexity separations in Koiran's weak model. *Theoretical Computer Science*, 133, 3–14, 1994.
5. I.M. Gelfand, M.M. Kapranov, A.V. Zelevinsky. Discriminants, resultants, and multidimensional determimants. Birkhäuser , 1994.
6. S.Å. Gustafson, K.O. Kortanek. Semi-infinte programming and applications. *Mathematical Programming: The State of the Art*, A. Bachem, M. Grötschel, B. Korte, eds., Springer, 132–157, 1983.
7. P. Koiran. A weak version of the Blum-Shub-Smale model. 34^{th} *Annual IEEE Symposium on Foundations of Computer Science*, 486–495, 1993.
8. V. Kreinovich, A.V. Lakeyev, J. Rohn, P. Kahl. Computational Complexity and Feasibility of Data Processing and Interval Computations. Kluwer, 1997.
9. V. Kreinovich, K. Meer. Complexity results for the range problem in interval arithmetic. In preparation.
10. M. Koshelev, L. Longpré, P. Taillibert. Optimal Enclusure of Quadratic Interval Functions. *Reliable Computing*, 4, 351–360, 1998.
11. A.V. Lakeyev, S.I. Noskov. A description of the set of solutions of a linear equation with interval defined operator and right-hand side. *Russian Acad. Sci. Dokl. Math.*, 47(3), 518–523, 1993.
12. K. Meer. On the complexity of quadratic programming in real number models of computation. *Theoretical Computer Science*, 133, 85–94, 1994.
13. K. Meer. On a refined analysis of some problems in interval arithmetic using real number complexity theory. *Reliable Computing*, 10(3), 209–225, 2004.
14. D.A. Plaisted. New NP-hard and NP-complete polynomial and integer divisibility problems. *Theoretical Computer Science*, 31, 125–138, 1984.
15. J. Rohn. Enclosing solutions of linear interval equations is NP-hard. *Computing*, 53, 365–368, 1094.
16. B. Sturmfels. Introduction to resultants. *Application of Computational Algebraic Geometry*, D.A. Cox B. Sturmfels (eds.), Proc. of Symposia in Applied Mathematics, Vol. 53, American Mathematical Society, 25–39, 1998
17. H. Woźniakowski: Why does information-based complexity use the real number model? Theoretical Computer Science 219, 451 – 465, 1999.

The Distributed Interval Geometric Machine Model*

Renata H.S. Reiser[1], Antônio C.R. Costa[1,2], and Graçaliz P. Dimuro[1]

[1] Escola de Informática
Universidade Católica de Pelotas
96010-000 Pelotas, Brazil
{reiser,rocha}@atlas.ucpel.tche.br
[2] Programa de Pós-Graduação em Computação
Universidade Federal do Rio Grande do Sul
90501-970, Porto Alegre, Brazil

Abstract. This paper presents a distributed version of the Interval Geometric Machine Model, called Distributed Interval Geometric Machine, whose inductive construction allows recursive definitions for interval algorithms involving possibly infinite distributed and synchronous parallel computations performed over array structures. In addition, the programming language $\mathcal{L}(\mathbb{D}_\infty)$ is extended to model the semantics of sample distributed algorithms applied to Interval Mathematics.

1 Introduction

Based on the set of transfinite ordinal numbers [10], an extension of the Interval Geometric Machine Model (IGM) [4,5], called Distributed Interval Geometric Machine (DIGM), is presented. This constructive model of distributed interval systems allows the modelling of non-determinism and two special types of parallelism: the temporal parallelism, with infinite memory and infinite processes defined over array structures, operating in a synchronized way; and the spatial parallelism, with a transfinite global memory shared by interval processes distributed in a enumerable set of Geometric Machines, temporarily synchronized. We take use of the advantages of Girard's Coherence Spaces [2] to obtain the domain-theoretic structure of the DIGM model. Following the methodology proposed by Scott[9], the coherence space of distributed processes, denoted by $\mathbb{D}_{\infty^2}$, is built over the coherence space of distributed elementary processes, which are single coherent subsets of tokens (actions labeled by positions of a geometric space) associated to an instant of computational time. The completion procedure of the space $\mathbb{D}_{\infty^2}$ ensures interpretations of spatial and temporal infinite computations. Each coherent set in $\mathbb{D}_{\infty^2}$ provides a description of the constructors (sequential or parallel products, deterministic or non-deterministic sums) of a distributed process and selects the machines used in its performance. The theoretical language $\mathcal{L}(\mathbb{D}_\infty)$ [7,6] induced by the interpretations obtained in the ordered structure $\mathbb{D}_\infty$ of the IGM model is extended to describe the interval arithmetic operations [3] involving the spatial and temporal recursive construction, based on coherence spaces and linear functions [2].

* Partially supported by CTINFO/CNPq and FAPERGS.

2 The Ordered Structure of the DIGM Model

In the following, we summarize the construction of the ordered structure of the DIGM model, including the following domains: $\mathbb{S}$ of spatially transfinite states, $\mathbb{B}$ of distributed Boolean tests and $\mathbb{D}_{\infty^2}$ of distributed processes. Firstly, basic concepts of coherence spaces and linear functions are considered, to obtain the ordered structure of the DIGM model, including the related morphisms.

2.1 Coherence Spaces and Linear Functions

A web $\mathbf{W} = (W, \approx_W)$ is a pair consisting of a set W with a symmetric and reflexive relation $\approx_W$, called coherence relation. A subset of this web with pairwise coherent elements is called a coherent subset. The collection of coherent subsets of the web $\mathbf{W}$, ordered by the inclusion relation, is called a *coherence space*, denoted by $\mathbb{W} \equiv (Coh(\mathbf{W}), \subseteq)$ [11].

Linear functions are continuous functions in the sense of Scott also satisfying the stability and linearity properties that assure the existence of least approximations in the image set [11]. Considering the coherence spaces $\mathbb{A}$ and $\mathbb{B}$, a linear function $f : \mathbb{A} \to \mathbb{B}$ is given by its *linear trace*, a subset of $A \times B$ given by $ltr(f) = \{(\alpha, \beta) \mid \beta \in f(\alpha)\}$. Let $\mathbf{A} \multimap \mathbf{B} = (A \times B, \approx_{\multimap})$ be the web of linear traces such that $(\alpha, \beta) \approx_{\multimap} (\alpha', \beta') \leftrightarrow ((\alpha \approx_{\mathbf{A}} \alpha' \to \beta \approx_{\mathbf{B}} \beta')$ and $(\beta = \beta' \to \alpha = \alpha'))$. The family of coherent subsets of the web $\mathbf{A} \multimap \mathbf{B}$, ordered by inclusion relation, defines the domain $\mathbb{A} \multimap \mathbb{B} \equiv (Coh(\mathbf{A} \multimap \mathbf{B}), \subseteq)$ of the linear traces of functions from $\mathbb{A}$ to $\mathbb{B}$.

2.2 The Coherence Space of Machine States

The machine states in DGM model are modeled as functions from memory positions to values [8]. When the memory values are taken in the set of real intervals we obtain the interval version of the DGM model. In a non-deterministic approach, machine states are modeled as families of deterministic machine states, with singletons modeling deterministic states.

The ordering of the natural numbers ($\mathbf{N}$), as dictated by their role as ordinals in [10], coincides with that in their original role as members of $\mathbf{N}$. The order type of the whole sequence $0, 1, 2, \ldots$ is usually denoted by ω and indicated here by $\mathbf{1}$. There follows its successor $\mathbf{1}0$, then $\mathbf{1}1$, and so on, whose order type is denoted by $\mathbf{2}$. Continuing on this fashion, we arrive at the following sequence:

$$0, 1, \ldots, \mathbf{1}0, \mathbf{1}1, \ldots, \mathbf{2}0, \mathbf{2}1, \ldots, \mathbf{m}0, \mathbf{m}1, \ldots$$

with order type ω^2, which is used to index its transfinite global memory. The DIGM memory is shared by synchronized processes distributed over an enumerable set of machines, each operating on a part $\mathbf{m}$ of such indexing. Let $\xi = (\xi_n)_{n \in \mathbf{N}}$ be a possible infinite enumerable and ordered set related to the multi-dimensional geometric space Ξ. In n-dimensional geometric space Ξ^n, $\xi = (\xi_0, \xi_1, \ldots, \xi_n)$ denotes the canonical basis and a n-dimensional vector is given by $(\alpha^{\xi_0}, \alpha^{\xi_1}, \ldots, \alpha^{\xi_n})$. When $m \in \mathbf{N}$, $\xi_n \in \xi$ and α^{ξ_n} denotes the value of the n-th coordinate of a vector in the geometric space, then

$(m, \alpha^{\xi_0}\alpha^{\xi_1}\ldots)$ denotes the memory position of a deterministic transfinite machine state, where $\mathbf{m}$ is the index of the machine that operates on that position.

The IGM model [5] was constructed over a geometric space given by the finite basic set $\xi = (\xi_0, \xi_1, \xi_2)$ and taking values in the set of real numbers. In this case, memory values are taken from the set of real intervals, whose elements ($a \in \mathbf{IR}$) are defined in the center-radius form ($(a_c, a_r) \in \mathbf{R} \times \mathbf{R}^+$) and placed in the IGM memory by the corresponding coordinates of the three-dimensional Euclidean geometric space ($(\alpha^{\xi_0}, \alpha^{\xi_1}, 0), (\alpha^{\xi_0}, \alpha^{\xi_1}, 1) \in \mathbf{R}^3$).

Let $\mathbb{IIQ} \equiv (Coh(\mathbf{IQ}), \subseteq)$ be the *Bi-structured Coherence Space of Rational Intervals* [1], representing the values that can be assigned to the variables. This domain is defined on the web $\mathbf{IQ} = (IQ, \approx_{IQ})$, given by the set of rational intervals with the coherence relation $p \approx_{IQ} q \leftrightarrow p \cap q \neq \emptyset$. In order to provide representation of the computable real numbers, it will be considered a restriction of $\mathbb{IIQ}$, called the domain of total objects of $\mathbb{IIQ}$ and denoted by $tot(\mathbb{IIQ})$. Thus, the ordered structure of the IGM memory is defined by the coherence space $\mathbb{R}^3 \multimap tot(\mathbb{IIQ})$, with $\mathbb{R}^3$ and $tot(\mathbb{IIQ})$ as flat domains representing the *memory values* and *memory positions*, respectively.

In the global transfinite memory of the distributed version of the IGM model, the real numbers a_c and a_r are labeled by the positions $(\mathbf{m}, \alpha^{\xi_0}, \alpha^{\xi_1}, 0)$ and $(\mathbf{m}, \alpha^{\xi_0}, \alpha^{\xi_1}, 1)$ belong to the geometric space $\mathbf{R}^3_\infty = \mathbf{N} \times \mathbf{R}^3$.

Taking $\mathbb{R}^3_\infty$ as the flat domain of memory positions:

Definition 1. *The coherence space modelling (transfinite) distributed deterministic states is defined by $\mathbb{R}^3_\infty \multimap \mathbb{IIQ}$. Let $\mathbf{S} = (Coh(\mathbb{R}^3_\infty \multimap \mathbb{IIQ}), \approx_S)$ be the web given by the set of all coherent subsets of $\mathbb{R}^3_\infty \multimap \mathbb{IIQ}$ together with the trivial (i.e., universal) coherence relation $\approx_S$. The coherence space $\mathbb{S} \equiv (Coh(\mathbf{S}), \subseteq)$ models the distributed non deterministic memory states in DIGM model, which is defined as the collection of all coherent subsets of $\mathbf{S}$, ordered by inclusion.*

In order to represent the deterministic sums of processes, we consider the set B of *distributed Boolean tests* related to the binary logic and assume the Coherence Space $\mathbb{B}$ of Boolean Tests as the set of all coherent subsets of tests of the discrete web $\mathbf{B} \equiv (B, =)$, ordered by the inclusion relation. The coherence space $\mathbb{S} \multimap \mathbb{B}$ models the distributed Boolean tests. Non-determinism enforces a non-traditional treatment of tests, in the same way that was done in [6] .

2.3 The Coherence Space of Processes

Now, we consider the main properties of the Coherence Space $\mathbb{D}_{\infty^2}$ of distributed processes in the DGM model[3], inductively constructed from the coherence space $\mathbb{D}_\infty$ introduced in [4]. Following the methodology proposed in [9], each level in the construction of $\mathbb{D}_{\infty^2}$ is identified by a subspace $\mathbb{D}_{\infty+m}$, which reconstructs all the objects from the level below it, preserving their properties and relations, and constructs the new objects specific of this level. The relationship between the levels is expressed by linear functions, called embedding and projection functions, interpreting constructors and destructors of processes, respectively. Based on the definition of an *elementary process* in

[3] The Coherence Spaces constructors used in the definition of $\mathbb{D}_{\infty^2}$ can be found in [11].

the IGM model [4], a *distributed elementary process* in the DIGM model is described by a transition between distributed deterministic memory states performed over a single memory position, in a single unit of computational time (uct). It is a function $\mathrm{d}^{(k)}$ satisfying:

Proposition 1. *Let* $\mathbb{A} \equiv [\mathbb{R}^3_\infty \multimap \mathbb{IIQ}] \multimap \mathbb{IIQ}$ *be the domain of the so-called computational actions. If* $\mathrm{d}, \mathrm{pr}^{(k)} \in \mathbb{A}$*, with* $\mathrm{pr}^{(k)}(s) = s(k)$ *then the next function* $\mathrm{d}^{(k)} \in [\mathbb{R}^3_\infty \rightarrow \mathbb{IIQ}]^2$ *is a linear function.*

$$\mathrm{d}^{(k)}(s)(i) = \begin{cases} \mathrm{pr}^{(i)}(s) & \text{if } i \neq k, \\ \mathrm{d}(s) & \text{if } i = k. \end{cases}$$

The collection of all linear functions $\mathrm{d}^{(k)}$ in $[\mathbb{R}^3_\infty \rightarrow \mathbb{IIQ}]^2$ satisfying the Prop. 1 is called the set of distributed elementary processes, whose elements can simultaneously read of distinct memory positions in the global shared memory (concurrent reading), but can only write to exactly one memory position (distributed exclusive writing). It means that in synchronous computations, each memory position is written to by exactly one process, in $1uct$.

In the global transfinite memory of the DIGM model, shared by an enumerable set of GM models, an elementary process can be conceived as a synchronization of a subset of distributed elementary processes, performed in the same memory position in all machine models.

The domain $\mathbb{D}_{\infty+m}$ realizes the set $\mathcal{D}_m$ of distributed elementary processes performed over the subset of transfinite memory position $\mathbf{R}^3_m = \{0, 1, \ldots, m\} \times \mathbf{R}^3 \subseteq \mathbf{R}^3_\infty$. The modeling of concurrency and the conflict of memory access between distributed processes in $\mathbb{D}_{\infty+m}$ is represented in the domains $\bar{\mathbb{D}}_{\infty+m}$ and $\bar{\mathbb{D}}^{\perp}_{\infty+m}$. Thus, taking the direct sum $\mathbb{P}_{\infty+m} = \mathbb{D}_{\infty+m} \coprod \bar{\mathbb{D}}_{\infty+m} \coprod \bar{\mathbb{D}}^{\perp}_{\infty+m}$, each level $\mathbb{D}_{\infty+m}$ is defined by the recursive domain equation

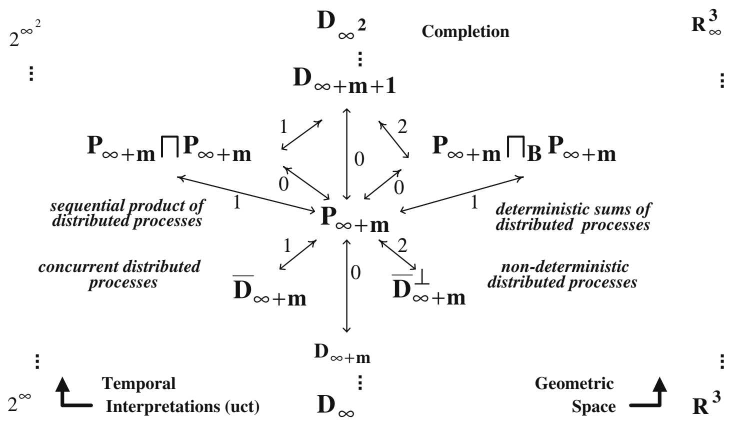
$$\mathbb{D}_{\infty+m+1} = \mathbb{P}_{\infty+m} \coprod (\mathbb{P}_{\infty+m} \prod \mathbb{P}_{\infty+m}) \coprod (\mathbb{P}_{\infty+m} \prod\nolimits_{\mathbb{B}} \mathbb{P}_{\infty+m}). \qquad (1)$$

The processes are represented in $\mathbb{D}_{\infty+m+1}$ as objects defined by coherent subsets of elementary processes labeled by positions of the geometric subspace $\mathbb{R}^3_{m+1}$ interpreting sequential products $(\mathbb{P}_{\infty+m} \prod \mathbb{P}_{\infty+m})$ or deterministic sums $(\mathbb{P}_{\infty+m} \prod_{\mathbb{B}} \mathbb{P}_{\infty+m})$ performed over parallel product or non-deterministic sums $(\mathbb{P}_{\infty+m})$ of distributed elementary processes constructed in the level below. Following the methodology resumed in Fig. 1, analogous descriptions can be obtained in the levels above, including the domain of distributed processes $\mathbb{D}_{\infty 2}$.

The objects in $\mathbb{D}_{\infty+m}$ are coherent subsets of indexed tokens. Each index (string) has a prefix modeling the relationship in the levels $\mathbb{D}_{\infty+m} - \mathbb{D}_{\infty+m+1}$, whose expression is obtained by concatenating finite substrings of two or three symbols. Let $\Sigma = \{0, 1, 2\} \times (\{0\} \bigcup (\{0, 1\} \times \{1, 2\}))$ be an alphabet and Σ^* as the set of finite words. Then an index is a string of the language

$$I = \{\alpha v; \beta w \mid \alpha v, \beta w \in \Sigma^* \Sigma_\infty,\ \alpha, \beta \in \Sigma^*,\ v, w \Sigma_\infty = \{: 00, : 001, : 002\}\}.$$

The leftmost symbol of a substring (α, β) indicates one of the constructors:
(0) for the inclusion of an element of the previous levels in the new domain,



Completion
sequential product of distributed processes
deterministic sums of distributed processes
concurrent distributed processes
non-deterministic distributed processes
Temporal Interpretations (uct)
Geometric Space

Fig. 1. Methodology of Construction of the $\mathbb{D}_{\infty^2}$

(1) indicating the parallel product of elements existing in the previous domains,
(2) denoting the non-deterministic sum of elements existing in the previous level.
In the level $\mathbb{P}_{\infty+m} - \mathbb{D}_{\infty+m+1}$, the second and third symbols, if present, mean:
(i) the first (02) or the second (12) summand in a deterministic sum, or
(ii) the first (01) or the second (11) term in a sequential product.
When the index is given by two symbols, the second one (0) is the inclusion.

The morphisms related to the indexes are projections and immersions, whose definition are obtained as extensions of the corresponding morphism presented in the levels $\mathbb{D}_m - \mathbb{P}_m$ and $\mathbb{P}_m - \mathbb{D}_{m+1}$, in the construction of the $\mathbb{D}_\infty$ [4].

The coherence space $\mathbb{D}_{\infty^2}$ of transfinite processes is introduced as a lest fixpoint of the equation (1). By the completion procedure, recursive processes can be represented in $\mathbb{D}_{\infty^2}$, without temporal and spatial constrained, including the immersion of coherent subsets from $\mathbb{D}_{\infty+m}$ to $\mathbb{D}_{\infty^2}$.This can be expressed by the suffix (w) of its indexed tokens $(\alpha v; \beta w)$, given by the following expressions: $(i) : 00 \equiv 00.00.00\ldots$ indicates finite or infinite processes (related to string αv);
$(ii) : 001 \equiv 001.001.001\ldots$ indicates transfinite sequential processes; and
$(iii) : 002 \equiv 002.002.002\ldots$ related to transfinite deterministic sums.

3 The Core Programming Language of $\mathcal{L}(\mathbb{D}_{\infty^2})$

Based on the interpretation of $\mathbb{D}_{\infty^2}$, we extend the programming language introduced in [6,7] for implementing parallel and sequential distributed algorithms in the DIGM model, in order to analyze temporal and spatial recursive construction. For that, an element (i, j) of an interval matrix stored in the memory of the DIGM model is given by a position pair $(\mathbf{a}, l, i, 0), (\mathbf{a}, l, i, 1) \in \mathbf{R}^3_\infty$, with the first coordinate indicating the machine $(\mathbf{a})$ that operates in such position, and the last one identifying its center (0) and radius (1) in the position $(\mathbf{a}, l, i) \in \mathbf{R}^2_\infty$.

The set of identifiers of the constant symbols is given by the union of the set I_P of symbols representing processes, where are included the elementary processes and the **skip** process, together with the set I_T of Boolean tests of $\mathbb{D}_{\infty^2}$. In addition, $F_{Op} = \{Id, \parallel, |, ;, +\}$ denotes the the constructors of distributed processes, which has Id as the symbol of the identity constructor, and $\|$, $|$ and $\cdot$ as the binary symbols representing the parallel product, sequential product and non deterministic sum, respectively. The deterministic sum is represented by $+ : I_P \times I_P \times I_T \rightarrow I_P$. Each one of these constructors admits a restriction related to a constrained written in the distributed shared memory. In this case, we take use of the index of the machine over that the constructor can be applied, e.g. $I_P/_{\{(\mathbf{0},l,i,u)\in\mathbf{R}^3_\infty\}} = \{Id_{\mathbf{0}}, \|_{\mathbf{0}}, \cdot_{\mathbf{0}}, |_{\mathbf{0}}, +_{\mathbf{0}}\} \subseteq I_P$

The set $\mathcal{L}(\mathbb{D}_{\infty^2})$ of expressions of the language of $\mathbb{D}^2_\infty$ includes variables and the above constant symbols. An expression of $\mathcal{L}(\mathbb{D}_{\infty^2})$ is a *representation* of a process of $\mathbb{D}_{\infty^2}$, and that the process is the *interpretation* of this expression.

Recursive equations can be used in $\mathcal{L}(\mathbb{D}_{\infty^2})$ to specify, through their fixpoints, expressions of distributed processes. Let $t_0, t_1, \ldots, t_n, t_{n+1}, \ldots b$ be elements of $\mathcal{L}(\mathbb{D}_{\infty^2})$. Thus, the distributed processes can be represented by finite, infinite or transfinite expressions of $\mathcal{L}(\mathbb{D}_{\infty^2})$ given by:

1. $*_{i=n}^{0}(t_i) = t_n * \ldots * t_0$ and $*_{i=0}^{n+1}(t_i) = t_0 * \ldots * t_n$;
2. $*_{n=0}^{\infty} t_n = t_0 * t_1 * \ \ldots \ * t_{n+1} * \ \ldots$;
3. $*_{m=0}^{\infty}(*_{n=0}^{\infty}) t_n = (t_{00} * t_{01} * \ldots * t_{0n} * \ldots) * \ldots * (t_{m0} * t_{m1} * \ldots * t_{mn} * \ldots) * \ldots$

We define the DIGM model and its computations following [8].

Definition 2. *The tuple of functions* $\mathcal{M} = (\mathcal{M}_{\mathbf{I}}, \mathcal{M}_{\mathbb{D}_{\infty}\mathbf{2}}, \mathcal{M}_{\mathbb{B}}, \mathcal{M}_{\mathbf{O}})$ *defines the DIGM model. The input and output values are taken in the set* IR *of real intervals, represented in the center-radius form* $[a_c, a_r]$*, and using* $s[i,j,k]$ *for* $s((i,j,k))$*:*

1. $\mathcal{M}_{\mathbf{I}} : \mathrm{IR} \rightarrow \mathbb{S}$ *is the* interval input function. *When* $\mathbf{I} = \{\mathbf{i}_{\mathbf{a}00}\}$*,*

$$\mathcal{M}_{\{\mathbf{i}_{\mathbf{a}00}\}}([a_c, a_r]) = \{s\}, s(\mathbf{a}, 0, 0, k) = \begin{cases} x_{a_c} \in \mathbb{IIQ}, \textit{ if } k = 0, \\ x_{a_r} \in \mathbb{IIQ}, \textit{ if } k = 1, \\ \emptyset, \qquad \textit{otherwise.} \end{cases}$$

where x_{a_c} *and* x_{a_r} *are the coherent sets of* $tot(\mathbb{IIQ})$ *that best approximate the real numbers* a_c *and* a_r*, see* [1].

2. $\mathcal{M}_{\mathbb{D}_{\infty^2}} : \mathcal{L}(\mathbb{D}_{\in\infty^2}) \multimap (\mathbb{S} \multimap \mathbb{S})$ *is the* program interpretation function*, such that* $\mathcal{M}_{\mathbb{D}_{\infty^2}}(p) = x$ *interprets the program* p *as the process* $x \in \mathbb{D}_{2\infty}$.

3. $\mathcal{M}_{\mathbb{B}} : I_T \multimap (\mathbb{S} \multimap \mathbb{B})$ *is the* test interpretation function*, such that* $\mathcal{M}_{\mathbb{B}}(\mathbf{b}) = t$ *interprets the symbol* $\mathbf{b}$ *as the test* $t \in \mathbb{B}$*, as it is defined in 2.2.*

4. $\mathcal{M}_{\mathbf{O}} : \mathbb{S} \rightarrow \mathrm{IR}$ *is the* output function. *When* $\mathbf{O} = \{\mathbf{o}_{ij}\}$*, then*

$$\mathcal{M}_{\{\mathbf{o}_{\mathbf{a}ij}\}}(\mathbf{s}) = \{[a_c, a_r] \mid s[\mathbf{a}, i, j, 0] = x_{a_c}, s[\mathbf{a}, i, j, 1] = x_{a_r} \in \mathbb{IIQ}, s \in \mathbf{s}\}.$$

The computation of a program p with an input data in results in the production of the output data $out = \mathcal{M}_{\mathbf{O}} \circ \mathcal{M}_{\mathbb{D}_\infty}(p) \circ \mathcal{M}_{\mathbf{I}}(in)$. An application of the language $\mathcal{L}(\mathbb{D}_{\infty^2})$, in the construction of sample interval algorithms for scalar multiplication and arithmetic operations, is presented.

4 Sample Interval Algorithms Expressed in $\mathcal{L}(\mathbb{D}_{\infty^2})$

Taking an interval in the center-radius form, $a_c, a_r, \alpha \in$ R, and the real functions $max, min, |\,|, \cdot, + : \mathrm{R}^2 \rightarrow \mathrm{R}$, the interval scalar multiplication $* : \mathrm{IR} \times \mathrm{R} \rightarrow \mathrm{IR}$ is defined by $\alpha * a = [\alpha \cdot a_c, |\alpha| \cdot b_c]$ and the interval arithmetic operations $\oplus, \ominus, \odot, \oslash$: $\mathrm{IR}^2 \rightarrow \mathrm{IR}$ and are given by

$$a \oplus b = [a_c + b_c, a_r + b_r] \quad a \odot b = 1/2 * [M_{max} + M_{min}, M_{max} - M_{min}],$$

$$a \ominus b = [a_c - b_c, a_r + b_r] \quad a \oslash b = \begin{cases} a \odot [\frac{b_c}{b_c^2 - b_r^2}, \frac{b_r}{b_c^2 - b_r^2}] & \text{if } |b_c| > b_r, \\ \text{undefined} & \text{if } |b_c| \leq b_r, \end{cases}$$

when $M = \{A, B, C, D\}$ and $A = a_c \cdot b_c + a_r \cdot b_c + a_c \cdot b_r + a_r \cdot b_r$, $B = a_c \cdot b_c - a_r \cdot b_c + a_c \cdot b_r - a_r \cdot b_r$, $C = a_c \cdot b_c + a_r \cdot b_c - a_c \cdot b_r - a_r \cdot b_r$ and $D = a_c \cdot b_c - a_r \cdot b_c - a_c \cdot b_r + a_r \cdot b_r$. Some expressions of elementary processes (assignment statements) and their corresponding interpretation in the domain $\mathbb{D}_{\infty^2}$ are given in Table 1. In order to simplify the denotation, consider $\alpha = s[\mathbf{b}, m, j, 0] \cdot s[\mathbf{c}, n, k, 0]$, $\beta = s[\mathbf{b}, m, j, 1] \cdot s[\mathbf{c}, n, k, 0]$, $\gamma = s[\mathbf{b}, m, j, 1] \cdot s[\mathbf{c}, n, k, 0]$ and $\theta = s[\mathbf{b}, m, j, 1] \cdot [\mathbf{c}, n, k, 1]$.

Table 1. Elementary processes and their domain interpretations

$\mathcal{L}(\mathbb{D}_{\infty^2})$	$\mathbb{D}_{\infty^2}$
$\mathtt{sum\_c}(\mathbf{a}, l, i; \mathbf{b}, m, j; \mathbf{c}, n, k) \equiv (s[\mathbf{a}, l, i, 0] := s[\mathbf{b}, m, j, 0] + s[\mathbf{c}, n, k, 0])$	$\{sum\_c^{(\mathbf{a},l,i,0)}_{:00;00}\}$
$\mathtt{sum\_r}(\mathbf{a}, l, i; \mathbf{b}, m, j; \mathbf{c}, n, k) \equiv (s[\mathbf{a}, l, i, 1] := s[\mathbf{b}, m, j, 1] + s[\mathbf{c}, n, k, 1])$	$\{sum\_c^{(\mathbf{a},l,i,1)}_{:00;00}\}$
$\mathtt{sub\_c}(\mathbf{a}, l, i; \mathbf{b}, m, j; \mathbf{c}, n, k) \equiv (s[\mathbf{a}, l, i, 0] := s[\mathbf{b}, m, j, 0] - s[\mathbf{c}, n, k, 0])$	$\{sub\_c^{(\mathbf{a},l,i,0)}_{:00;00}\}$
$\mathtt{sc\_c}(\mathbf{a}, l, i; \mathbf{b}, m, j; \alpha) \equiv (s[\mathbf{a}, l, i, 0] := |\alpha| \cdot s[\mathbf{a}, m, j, 0]$	$\{esc\_c^{(\mathbf{a},l,i,0)}_{:00;00}\}$
$\mathtt{sc\_r}(\mathbf{a}, l, i; \mathbf{b}, m, j; \alpha) \equiv (s[\mathbf{a}, l, i, 1] := \alpha \cdot s[\mathbf{b}, m, j, 0]$	$\{esc\_c^{(\mathbf{a},l,i,0)}_{:00;00}\}$
$\mathtt{p\_2}(\mathbf{a}, l, i, u; \mathbf{b}, m, j; \mathbf{c}, n, k) \equiv (s[\mathbf{a}, l, i, u+2] := \alpha + \beta + \gamma + \theta)$	$\{p\_2^{(\mathbf{a},l,i,u+2)}_{:00;00}\}$
$\mathtt{p\_3}(\mathbf{a}, l, i, u; \mathbf{b}, m, j; \mathbf{c}, n, k) \equiv (s[\mathbf{a}, l, i, u+3] := \alpha - \beta + \gamma + \theta)$	$\{p\_3^{(\mathbf{a},l,i,u+3)}_{:00;00}\}$
$\mathtt{p\_4}(\mathbf{a}, l, i, u; \mathbf{b}, m, j; \mathbf{c}, n, k) \equiv (s[\mathbf{a}, l, i, u+4] := \alpha + \beta - \gamma + \theta)$	$\{p\_4^{(\mathbf{a},l,i,u+4)}_{:00;00}\}$
$\mathtt{p\_5}(\mathbf{a}, l, i, u; \mathbf{b}, m, j; \mathbf{c}, n, k) \equiv (s[\mathbf{a}, l, i, u+5] := \alpha - \beta - \gamma + \theta)$	$\{p\_5^{(\mathbf{a},l,i,u+5)}_{:00;00}\}$
$\mathtt{zero}(\mathbf{a}, l, i, u) \equiv (s[\mathbf{a}, l, i, u] := 0)$	$\{zero^{(\mathbf{a},l,i,u)}_{:00;00}\}$

Based on the above assignment statements expressing distributed elementary processes, we illustrate the representation of some compound distributed (non elementary) processes in DIGM. Let $F_{(10)} : \mathbb{D}_{\infty^2} \sqcap \mathbb{D}_{\infty^2} \rightarrow \mathbb{D}_{\infty^2}$ be the parallel product operator of $\mathbb{D}_{\infty^2}$, represented in $\mathcal{L}(\mathbb{D}_{\infty^2})$ by the symbol $\|$. In addition, $F_{\mathbf{m}(10)} : \mathbb{D}_{\infty+\mathbf{m}} \sqcap \mathbb{D}_{\infty+\mathbf{m}} \rightarrow \mathbb{D}_{\infty+\mathbf{m}}$ is the correspond restriction.

(1) $\mathtt{Zero}(\mathbf{a}, l, i, u) \equiv (\mathtt{zero}(\mathbf{a}, l, i, u) \parallel \mathtt{Zero}(\mathbf{a}+1, l, i, u))$
$\mathtt{zero}(\mathbf{a}, l, i, u) \equiv (\mathtt{zero}(\mathbf{a}, l, i, u) \parallel_{\mathbf{a}} \mathtt{Zero}(\mathbf{a}, l+1, i, u))$
The expressions $\mathtt{Zero}(\mathbf{0}, 0, 0, 0)$ and $\mathtt{zero}(\mathbf{0}, 0, 0, 0)$ represent the processes of initialization performed over the distributed memory in $1\,uct$, but they are recursively defined by iterations applied to distinct enumerable subsets of the memory position. The action executed by the first process change the memory values of the first position in all blocks. The action of the second one is performed over the first column and restricted to the first block. When $x_0 = \{\overline{zero^{(\mathbf{0},l,i,1)}}_{10:00}\}$, their related interpretation

$$Z = \bigcup_{\mathbf{a}} \{\overline{zero^{(\mathbf{a},l,i,0)}_{:00;10:00}}\} \in \mathbb{D}_{\infty^2} \qquad z = \bigcup_{l} \{\overline{zero^{(\mathbf{a},l,i,0)}_{10:00;00}}\} \in \mathbb{D}_{\infty^2}$$

can be computed as a fixpoint of the spatial recursive equation

$$F_{(10)}(\{\overline{zero^{(\mathbf{a+1},l,i,0)}_{00;10:00}}\} \sqcap x_a) = x_{a+1} \qquad F_{\mathbf{a}(10)}(\{\overline{zero^{(\mathbf{a},l+1,i,0)}_{10:00;00}}\} \sqcap x_l) = x_{l+1},$$

Analogous construction can be obtained to the other constructors. Some compound distributed processes related to addition are presented below.

(2) $\mathtt{sum}(\mathbf{a}, l, i, m, j, n, k) \equiv (\mathtt{sum\_c}(\mathbf{a}, l, i; \mathbf{a}, m, j; \mathbf{a}, n, k) \parallel_a \mathtt{sum\_r}(\mathbf{a}, l, i; \mathbf{a}, m, j; \mathbf{a}, n, k)$
This expression is the sum of two intervals, labeled by reference positions $(\mathbf{a}, m, j)$ and $(\mathbf{a}, n, k)$, with the result placed in the $(\mathbf{a}, l, i)$ position in the $\mathbf{a} - th$ block of the global memory. The sum of the centers and the sum of the radii performed in parallel, in $1uct$.

(3) $\begin{cases} \mathtt{sum\_row}(\mathbf{a}, l, m, n, i) \equiv (\mathtt{sum}(\mathbf{a}, l, i, m, i, n, i) \parallel_a \mathtt{sum\_row}(\mathbf{a}, l, m, n, i-1)) \\ \mathtt{sum\_row}(\mathbf{a}, l, m, n, i) \equiv \mathtt{Skip} \end{cases}$
$\mathtt{Sum\_row}(\mathbf{a}, l, m, n, i) \equiv (\mathtt{sum\_row}(\mathbf{a}, l, m, n, i) \parallel \mathtt{Sum\_row}(\mathbf{a}+1, l, m, n, i)$
When $s[\mathbf{a}, l, i+1] := 0$, the recursive expression $\mathtt{Sum\_row}(\mathbf{0}, l, m, n, 10)$ represents the process that executes, in parallel, the addition of the first ten intervals stored in the m-th and n-th rows, and assigns the result to the corresponding positions in the l-th row. This action is simultaneously performed over all blocks in the DIGM memory, in $1uct$.

(4) $\begin{cases} \mathtt{Sum\_Row}(\mathbf{a}, l, 0) \equiv \mathtt{sum}(\mathbf{a}, l, 0; \mathbf{a}, l, 0; \mathbf{a}, l, 1), \\ \mathtt{Sum\_Row}(\mathbf{a}, l, i) \equiv (\mathtt{sum}(\mathbf{a}, l, i, l, i, l, i+1) \cdot_a \mathtt{Sum\_Row}(\mathbf{a}, l, i-1)). \end{cases}$
The above algorithm is a temporal recursive process related to sequential product, whose time performance is related to i. Thus, $\mathtt{Sum\_Row}(\mathbf{a}, l, i)$ accumulates the addition of the first $i+1$-th intervals stored in the $l - th$ row. The result is obtained in the last iteration and it is placed in the first position of the corresponding row. The synchronous performance of this process, over the first $\mathbf{a} - th$ blocks of the transfinite memory, can be expressed by:
$\mathtt{R}(\mathbf{0}, l, i) \equiv \mathbf{skip}, \quad \mathtt{S}(\mathbf{a}, l, i) \equiv \mathtt{Sum\_Row}(\mathbf{a}, l, i) \parallel \mathtt{R}(\mathbf{a}-1, l, i)).$

(5) $\mathtt{S}(\mathbf{a}, l, ; \mathbf{b}, j; \mathbf{c}, k) \equiv (\mathtt{sum\_c}(\mathbf{a}, l, k+1; \mathbf{b}, l, j; \mathbf{c}, j, k) \parallel_a \mathtt{sum\_r}(\mathbf{a}, l, k+1; \mathbf{b}, l, j; \mathbf{c}, j, k)).$
$\mathtt{O}(\mathbf{a}, \mathbf{b}, \mathbf{c}, l, j, k) \equiv (\mathtt{S}(\mathbf{a}, l, ; \mathbf{b}, j; \mathbf{c}, k) \cdot \mathtt{sum}(\mathbf{a}, l, k, l, k, l, k+1)) \parallel_a \mathtt{O}(\mathbf{a}, \mathbf{b}, \mathbf{c}, l, j+1, k).$
When $\mathtt{inic}(\mathbf{a}, l, k) \equiv (\mathtt{zero}(\mathbf{a}, l, k+1, 0) \parallel_a \mathtt{zero}(\mathbf{a}, l, k+1, 1)) \parallel_a s[a, l, k, 0] := 0,$

the expression $\texttt{O}(\mathbf{a},\mathbf{b},\mathbf{c},l,j,k)$ interprets an infinite distributed process performing the accumulated addition between the intervals of the l-th row, in the **b**-th block, and the correspond elements of the column $k-th$, in the **c**-th block. The result is placed in the (l,k)-th position of the **a**-th block.

Now, other expressions involving scalar product are considered.

(6) $\texttt{sc}(\mathbf{a},l,i;\mathbf{b},m,j;\alpha) \equiv (\texttt{sc\_c}(\mathbf{a},l,i;\mathbf{b},m,j;\alpha) \parallel_a \texttt{sc\_r}(\mathbf{a},l,i;\mathbf{b},m,j;\alpha)$ This expression represents the scalar multiplication, where the real operations related to the centers and the radiuses are performed in parallel, in $1uct$.

(7) $$\begin{cases} \texttt{rot}(\mathbf{a},l,i,p) \equiv (\texttt{sc}(\mathbf{a},l,i-1;\mathbf{a},l,i;1) \parallel_a \texttt{rot}(\mathbf{a},l,i-1,p+1)) \\ \texttt{rot}(\mathbf{a},l,0,p) \equiv (\texttt{sc}(\mathbf{a},l,p;\mathbf{a},l,i;1) \end{cases}$$

The expression $\texttt{Rot}(\mathbf{a},l,i,p) \equiv \texttt{rot}(\mathbf{a},l,i,p) \parallel_a \texttt{Rot}(\mathbf{a}+1,l,i,p)$ represents a distributed process performing a periodic rotation in all transfinite memory.

(8) $$\begin{cases} \texttt{sign}(\mathbf{a},l,m,i) \equiv (\texttt{sc}(\mathbf{a},l,i;\mathbf{a},m,i;(-1)^i) \parallel_a \texttt{sign}(\mathbf{a},l,m,i-1)) \\ \texttt{sign}(\mathbf{a},l,m,0) \equiv \mathbf{skip}) \end{cases}$$

Based on (8), the sign in the alternative transfinite memory position can be changed by a distributed process, whose expression in $\mathcal{L}(\mathbb{D}_{\infty^2})$ is given by
$\texttt{Sign}(\mathbf{a},l,i,p) \equiv \texttt{sign}(\mathbf{a},l,i,p) \parallel_a \texttt{Sign}(\mathbf{a}+1,l,i,p)$.

In the following, multiplication between interval matrices is presented.

(9) $$\begin{cases} \texttt{Max}(\mathbf{a},l,i,0) := \texttt{skip} \\ \texttt{Max}(\mathbf{a},l,i,u) := \texttt{maxi}(\mathbf{a},l,i,u-1) \cdot_a \texttt{Max}(\mathbf{a},l,i,u-1), \end{cases}$$

This is an example of a temporal recursive process, which is obtained by iterating the sequential product and related to the maximal elements, when

$$\begin{cases} \texttt{maxi}(\mathbf{a},l,i,2) \equiv \texttt{skip}, \\ \texttt{maxi}(\mathbf{a},l,i,u) \equiv (s[\mathbf{a},l,i,u+2] := max(s[\mathbf{b},m,j,u+2], s[\mathbf{c},n,k,u+3]) \end{cases}$$

The same construction can be obtained if we consider the minimum operator.

(10) $\texttt{p}(\mathbf{a},l,m,n,i,j,k) \equiv (\texttt{p\_par}(\mathbf{a},l,m,n,i,j,k) \cdot_a (\texttt{p\_c}(\mathbf{a},l,i) \parallel_a \texttt{p\_r}(\mathbf{a},l,i))),$
is a parallel product performing the multiplication of an interval placed in the (m,j)-th memory position by another one in the (n,k)-th memory position, and the result is placed in the (l,i)-th position, in the **a**-th block of shared memory. In this case, the definition of the other operations are presented in the following.

$$\begin{aligned} \texttt{p\_par}(\mathbf{a},\mathbf{b},\mathbf{c},l,m,n,i,j,k) \equiv{} & \texttt{p\_2}(\mathbf{a},\mathbf{b},\mathbf{c},l,m,n,i,j,k,2) \parallel_a \\ & \texttt{p\_3}(\mathbf{a},\mathbf{b},\mathbf{c},l,m,n,i,j,k,2) \parallel_a \\ & \texttt{p\_4}(\mathbf{a},\mathbf{b},\mathbf{c},l,m,n,i,j,k,2) \parallel \\ {}_a \quad & \texttt{p\_5}(\mathbf{a},\mathbf{b},\mathbf{c},l,m,n,i,j,k,2) \\ \texttt{p\_c}(\mathbf{a},l,i) \equiv{} & (s[\mathbf{a},l,i,0] := (\texttt{Max}(\mathbf{a},l,i,5) + \texttt{Min}(\mathbf{a},l,i,5))/2) \\ \texttt{p\_r}(\mathbf{a},l,i) \equiv{} & (s[\mathbf{a},l,i,1] := (\texttt{Max}(\mathbf{a},l,i,5) - \texttt{Min}(\mathbf{a},l,i,5))/2) \end{aligned}$$

(11) $$\begin{cases} \texttt{pd\_row}(\mathbf{a},n,0,n,i) \equiv (\texttt{p}(\mathbf{a},n,0,n,i,i,i) \cdot_a R(\mathbf{a},n,i)) \parallel_a \texttt{pd\_row}(\mathbf{a},n,0,n,i-1) \\ \texttt{pd\_row}(\mathbf{a},n,0,n,0) \equiv \texttt{p}(\mathbf{a},n,0,n,0,0,0) \cdot_a R(\mathbf{a},n,0) \end{cases}$$

When $R(\mathbf{a},n,0) \equiv (s[\mathbf{a},l,i+1] := 0 \cdot sum(\mathbf{a},l,l,l,i,i,i+1)(4)$, the expression (11) interprets the synchronous product of interval arrays of i-th dimension, whose position is related to the a-th block of shared memory, in the 0-th and n-th rows. Thus, the summation is placed in the position $(\mathbf{a},n,0)$ and the iteration of the process in (11) is obtained by the recursive expression

$$\begin{cases} \texttt{Prod\_row}(\mathbf{a}, n, i) \equiv \texttt{pd\_row}(\mathbf{a}, n, 0, n, i) \parallel \texttt{Prod\_row}(\mathbf{a}, n-1, i) \\ \texttt{Prod\_row}(\mathbf{a}, 0, i) \equiv \texttt{pd\_row}(\mathbf{a}, 0, 0, 0, i). \end{cases}$$

The above algorithms $(10 - 11)$ compound the next process, whose expression provide interpretation for the multiplication of interval matrices related to synchronous product performed by distributed processes:

$\texttt{PRow}(\mathbf{a}, n, i) \equiv \texttt{Prod\_row}(\mathbf{a}, n, i) \parallel \texttt{PRow}(\mathbf{a}-1, n, i)$, $\texttt{PRow}(\mathbf{0}, n, i) \equiv \texttt{pd\_row}(\mathbf{0}, n, i)$.

5 Conclusions

In this work we described the main characteristics of the DIGM model. Sample examples of distributed programs are written in the programming language extracted from $\mathbb{D}_{\infty^2}$. The visual programming environment, designed to support the programming in the theoretical DIGM is work in progress. In this environment, various kinds of parallel computations involving array structures, such as array computations and cellular automata, will be possible. This language allows (spacial and temporal) semantic specification of a process in a bi-dimensional way. In addition, the visual representation allows simulations performed by parallel processes in a intuitive way.

References

1. G. P. Dimuro, A. C. R. Costa and D. M. Claudio, *A Coherence Space of Rational Intervals for a Construction of IR*, Reliable Computing **6**(2), (2000), 139–178.
2. J. -Y. Girard, *Linear logic*, Theoretical Computer Science **1** (1987), 187–212.
3. R. E. Moore, *Methods and Applications of Interval Analysis*, SIAM Publ., Philadelphia, 1979.
4. R. H. S. Reiser, A. C. R. Costa and G. P. Dimuro, *First steps in the construction of the Geometric Machine*, TEMA. RJ: SBMAC, **3**(1), (2002), 183–192.
5. R. H. S. Reiser, A. C. R. Costa and G. P. Dimuro, *The Interval Geometric Machine*, Numerical Algorithms, Dordrecht: Kluwer, **37** (2004), 357–366.
6. R. H. S. Reiser, A. C. R. Costa and G. P. Dimuro, *A Programming Language for the Interval Geometric Machine Model*, Electronic Notes in Theoretical Computer Science **84** (2003), 1–12.
7. R. H. S. Reiser, A. C. R. Costa and G. P. Dimuro, *Programming in the Geometric Machine*, Frontiers in Artificial Intelligence and Its Applications, Amsterdam: IOS Press, **101** (2003), 95–102.
8. D. Scott, *Some Definitional Suggestions for Automata Theory*, Journal of Computer and System Sciences, New York, **1** (1967), 187–212.
9. D. Scott, *The lattice of flow diagrams*, Lecture Notes. Berlin: Springer Verlag, **188**(1971), 311–372.
10. R. R. Stoll, Set Theory and Logic. New York: Dover Publication Inc. 1961. 474 p.
11. A. S. Troelstra, *Lectures on Linear Logic*, Lecture Notes. Stanford: CSLI/Leland Stanford Junior University, **29** (1992).

New Algorithms for Statistical Analysis of Interval Data

Gang Xiang, Scott A. Starks, Vladik Kreinovich, and Luc Longpré

NASA Pan-American Center for Earth and Environmental Studies (PACES)
University of Texas, El Paso, TX 79968, USA
vladik@cs.utep.edu

Abstract. It is known that in general, statistical analysis of interval data is an NP-hard problem: even computing the variance of interval data is, in general, NP-hard. Until now, only one case was known for which a feasible algorithm can compute the variance of interval data: the case when all the measurements are accurate enough – so that even after the measurement, we can distinguish between different measured values $\widetilde{x}_i$. In this paper, we describe several new cases in which feasible algorithms are possible – e.g., the case when all the measurements are done by using the same (not necessarily very accurate) measurement instrument – or at least a limited number of different measuring instruments.

1 Introduction

Once we have several results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of measuring some physical quantity – e.g., the amount of pollution in a lake – traditional statistical data processing starts with computing the sample average $E = E(\widetilde{x}_1, \ldots, \widetilde{x}_n)$, the sample median $M = M(\widetilde{x}_1, \ldots, \widetilde{x}_n)$, and the sample variance $V = V(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ of these results. For example,

$$E(\widetilde{x}_1, \ldots, \widetilde{x}_n) = \frac{1}{n} \sum_{i=1}^{n} \widetilde{x}_i.$$

The values $\widetilde{x}_i$ come from measurements, and measurements are never 100% accurate. In many real-life situations, the only information about the corresponding measurement errors is the upper bound Δ_i on the absolute value of the measurement error. As a result, the only information we have about the actual value x_i of each measured quantity is that x_i belongs to the interval $\mathbf{x}_i \stackrel{\text{def}}{=} [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$.

For interval data, instead of the exact values of E, M, and V, it is desirable to get the intervals $\mathbf{E}$, $\mathbf{M}$, and $\mathbf{V}$ of possible values, intervals formed by all possible values of E (correspondingly, M or V) when each x_i takes values from the interval $\mathbf{x}_i$.

Computing $\mathbf{E} = [\underline{E}, \overline{E}]$ and $\mathbf{M} = [\underline{M}, \overline{M}]$ is straightforward: indeed, both the sample average E and the sample median M are (non-strictly) increasing functions of the variables $x_1, \ldots, x_n$. So, the smallest possible value $\underline{E}$ (correspondingly, $\underline{M}$) is attained when we take the smallest possible values $\underline{x}_1, \ldots, \underline{x}_n$ from the corresponding intervals; similarly, the largest possible value $\overline{E}$ (correspondingly, $\overline{M}$) is attained when we take the largest possible values $\overline{x}_1, \ldots, \overline{x}_n$ from the corresponding intervals. Thus, $\underline{E} = E(\underline{x}_1, \ldots, \underline{x}_n)$, $\overline{E} = E(\overline{x}_1, \ldots, \overline{x}_n)$, $\underline{M} = M(\underline{x}_1, \ldots, \underline{x}_n)$, and $\overline{M} = M(\overline{x}_1, \ldots, \overline{x}_n)$.

On the other hand, computing the exact range $\mathbf{V} = [\underline{V}, \overline{V}]$ of V turns out to be an NP-hard problem; specifically, computing the upper endpoint $\overline{V}$ is NP-hard (see, e.g., [2]).

It is worth mentioning that computing the lower endpoint $\underline{V}$ is feasible; in [3], we show that it can done in time $O(n \cdot \log(n))$.

In the same paper [2] in which we prove that computing $\overline{V}$ is, in general, NP-hard, we also show that in the case when the measuring instruments are accurate enough – so that even after the measurements, we can distinguish between different measured values $\widetilde{x}_i$ (e.g, if the corresponding intervals $\mathbf{x}_i$ do not intersect) – we can compute $\overline{V}$ (hence, $\mathbf{V}$) in feasible time (actually, quadratic time).

In some practical examples, the measurement instruments are indeed very accurate, but in many other practical cases, their accuracy may be much lower – so the algorithm from [2] is not applicable.

In this paper, we describe new practically useful cases when we can compute $\mathbf{V}$ by a feasible (polynomial-time) algorithm.

The first case is when all the measurements are made by the same measuring instrument or by similar measurement instruments. In this case, none of two input intervals $\mathbf{x}_i$ is a proper subset of one another, and as a result, we can find the exact range $\mathbf{V}$ in time $O(n \cdot \log(n))$.

The second case is when instead of a single type of measuring instruments, we use a limited number ($m > 1$) of different types of measuring instruments. It turns out that in this case, we can compute $\mathbf{V}$ in polynomial time $O(n^{m+1})$.

The third case is related to privacy in statistical databases; see details below.

2 First Case: Measurements by Same Measuring Instrument

In the proof that computing variance is NP-hard (given in [2]), we used interval data in which some intervals are proper subintervals of others: $\mathbf{x}_i \subset \mathbf{x}_j$ (and $\mathbf{x}_i \neq \mathbf{x}_j$).

From the practical viewpoint, this situation makes perfect sense: the interval data may contain values measurement by more accurate measuring instruments – that produce narrower intervals $\mathbf{x}_i$ – and by less accurate measurement instruments – that produce wider intervals $\mathbf{x}_j$. When we measure the same value $x_i = x_j$, once with an accurate measurement instrument, and then with a less accurate instrument, then it is quite possible that the wider interval corresponding to the less accurate measurement properly contains the narrower interval corresponding to the more accurate instrument.

Similarly, if we measure close values $x_i \approx x_j$, it is quite possible that the wider interval coming from the less accurate instrument contains the narrower interval coming from the more accurate instrument.

In view of the above analysis, a natural way to avoid such difficult-to-compute situations is to restrict ourselves to situations when all the measurement are done with the same measuring instrument.

For a single measuring instrument, it is not very probable that in two different measurements, we get two intervals in which one is a proper subinterval of the other.

Let us show that in this case, we have a feasible algorithm for computing $\overline{V}$. For each interval $\mathbf{x} = [\underline{x}, \overline{x}]$, we will denote its half-width $(\overline{x} - \underline{x})/2$ by Δ, and its midpoint $(\underline{x} + \overline{x})/2$ by $\widetilde{x}$.

Definition 1. *By an* interval data*, we mean a finite set of intervals* $\mathbf{x}_1, \ldots, \mathbf{x}_n$.

Definition 2. *For n real numbers $x_1, \ldots, x_n$, their* variance $V(x_1, \ldots, x_n)$ *is defined in the standard way – as* $V \overset{\text{def}}{=} \frac{1}{n} \cdot \sum\limits_{i=1}^{n} x_i^2 - E^2$*, where* $E \overset{\text{def}}{=} \frac{1}{n} \cdot \sum\limits_{i=1}^{n} x_i$.

Definition 3. *By the* interval variance $\mathbf{V}$ *of the interval data, we mean the interval* $\mathbf{V} \overset{\text{def}}{=} \{V(x_1, \ldots, x_n) \,|\, x_i \in \mathbf{x}_i\}$ *filled by the values $V(x_1, \ldots, x_n)$ corresponding to different $x_i \in \mathbf{x}_i$.*

Theorem 1. *There exists an algorithm that computes the variance $\mathbf{V}$ of the interval data in time $O(n \cdot \log(n))$ for all the cases in which no element of the interval data is a subset of another element.*

Proof. In order to compute the interval $\mathbf{V}$, we must compute both endpoints $\underline{V}$ and $\overline{V}$ of this interval.

The proof of the theorem consists of three parts:

- in Part A, we will mention that the algorithm for computing $\underline{V}$ in time $O(n \cdot \log(n))$ is already known;
- in Part B, we describe a new algorithm for computing $\overline{v}$;
- in Part C, we prove that the new algorithm is correct and has the desired complexity.

A. Algorithm for computing $\underline{V}$ with desired time complexity is already known. The algorithm for computing $\underline{V}$ in time $O(n \cdot \log(n))$ is described in [4].

B. New algorithm for computing $\overline{V}$: description. The proposed algorithm for computing $\overline{V}$ is as follows:

- First, we sort n intervals $\mathbf{x}_i$ in lexicographic order:

$$\mathbf{x}_1 \leq_{\text{lex}} \mathbf{x}_2 \leq_{\text{lex}} \cdots \leq_{\text{lex}} \mathbf{x}_n,$$

  where $[\underline{a}, \overline{a}] \leq_{\text{lex}} [\underline{b}, \overline{b}]$ if and only if either $\underline{a} < \underline{b}$, or $\underline{a} = \underline{b}$ and $\overline{a} \leq \overline{b}$.
- Second, we use bisection to find the value k ($1 \leq k \leq n$) for which the following two inequalities hold:

$$\widetilde{x}_k + \frac{1}{n} \cdot \sum_{i=1}^{k-1} \Delta_i \leq \frac{1}{n} \cdot \sum_{i=k+1}^{n} \Delta_i + \frac{1}{n} \cdot \sum_{i=1}^{n} \widetilde{x}_i; \tag{2.1}$$

$$\widetilde{x}_{k+1} + \frac{1}{n} \cdot \sum_{i=1}^{k} \Delta_i \geq \frac{1}{n} \cdot \sum_{i=k+2}^{n} \Delta_i + \frac{1}{n} \cdot \sum_{i=1}^{n} \widetilde{x}_i. \tag{2.2}$$

  At each iteration of this bisection, we have an interval $[k^-, k^+]$ that is guaranteed to contain k. In the beginning, $k^- = 1$ and $k^+ = n$. At each stage, we compute the midpoint $k_{\text{mid}} = \lfloor (k^- + k^+)/2 \rfloor$, and check both inequalities (2.1) and (2.2) for $k = k_{\text{mid}}$. Then:

  - If both inequalities (2.1) and (2.2) hold for his k, this means that we have found the desired k.
  - If (2.1) holds but (2.2) does not hold, this means that the desired value k is larger than k_{mid}, so we keep k^+ and replace k^- with $k_{\mathrm{mid}} + 1$.
  - If (2.2) holds but (2.1) does not hold, this means that the desired value k is smaller than k_{mid}, so we keep k^- and replace k^+ with $k_{\mathrm{mid}} - 1$.
- Once k is found, we compute

$$V_k \stackrel{\text{def}}{=} \frac{1}{n} \cdot \sum_{i=1}^{k} \underline{x}_i^2 + \frac{1}{n} \cdot \sum_{i=k+1}^{n} \overline{x}_i^2 - \left(\frac{1}{n} \sum_{i=1}^{k} \underline{x}_i + \frac{1}{n} \cdot \sum_{i=k+1}^{n} \overline{x}_i \right)^2 . \tag{2.3}$$

  This is the desired value $\overline{V}$.

C. Proof of correctness and complexity. Let us prove that this algorithm indeed produces the correct result and indeed requires time $O(n \cdot \log(n))$.

1°. Let us first prove that if no element of the interval data is a subset of another element, then, after we sort these elements in lexicographic order, both the lower endpoints $\underline{x}_i$ and the upper endpoints $\overline{x}_i$ are sorted in non-decreasing order: $\underline{x}_i \le \underline{x}_{i+1}$ and $\overline{x}_i \le \overline{x}_{i+1}$.

Indeed, by definition of a lexicographic order, we always have $\underline{x}_i \le \underline{x}_{i+1}$. If $\underline{x}_i = \underline{x}_{i+1}$, then, by definition of the lexicographic order, we have $\overline{x}_i \le \overline{x}_{i+1}$. If $\underline{x}_i < \underline{x}_{i+1}$, then we cannot have $\overline{x}_i \ge \overline{x}_{i+1}$ – otherwise, we would have $\mathbf{x}_{i+1} \subset \mathbf{x}_i$ – hence $\overline{x}_i < \overline{x}_{i+1}$. The statement is proven.

It is known that sorting requires time $O(n \cdot \log(n))$; see, e.g., [1].

In the following text, we will assume that the sequence of intervals has been sorted in this manner.

2°. Let us now prove that the desired maximum of the variance V is attained when each variable x_i is at one of the endpoints of the corresponding interval $\mathbf{x}_i$.

Indeed, if the maximum is attained in the interior point of this interval, this would means that in this point, $\partial V / \partial x_i = 0$ and $\partial^2 V / \partial x_i^2 \le 0$. For variance, $\partial V / \partial x_i = (2/n) \cdot (x_i - E)$, so $\partial^2 V / \partial x_i^2 = (2/n) \cdot (1 - 1/n) > 0$ – hence maximum cannot be inside.

3°. Let us show the maximum is attained at a vector

$$x = (\underline{x}_1, \ldots, \underline{x}_k, \overline{x}_{k+1}, \ldots, \overline{x}_n) \tag{2.4}$$

in which we first have lower endpoints and then upper endpoints.

What we need to prove is that there exists a maximizing vector in which, once we have an upper endpoint, what follows will also be an upper endpoint, i.e., in which we cannot have $x_k = \overline{x}_k > \underline{x}_k$ and $x_{k+1} = \underline{x}_{k+1} < \overline{x}_{k+1}$.

For that, let us start with a maximizing vector in which this property does not hold, i.e., in which $x_k = \overline{x}_k > \underline{x}_k$ and $x_{k+1} = \underline{x}_{k+1} < \overline{x}_{k+1}$ for some k. Based on this vector, we will now construct a different maximizing vector with the desired property. For that, let us consider two cases: $\Delta_k < \Delta_{k+1}$ and $\Delta_k \geq \Delta_{k+1}$, where $\Delta_i \overset{\text{def}}{=} (\overline{x}_i - \underline{x}_i)/2$ is the half-width of the interval $\mathbf{x}_i$.

In the first case, let us replace $\overline{x}_k = \underline{x}_k + 2\Delta_k$ with $\underline{x}_k$, and $\underline{x}_{k+1}$ with $\underline{x}_{k+1} + 2\Delta_k$ (since $\Delta_k < \Delta_{k+1}$, this new value is $< \overline{x}_{k+1}$). Here, the average E remains the same, so the only difference between the new value V' of the variance and its old value V comes from the change in terms x_k^2 and x_{k+1}^2. In other words,

$$V' - V = \frac{1}{n} \cdot ((\underline{x}_{k+1} + 2\Delta_k)^2 - \underline{x}_{k+1}^2) - \frac{1}{n} \cdot ((\underline{x}_k + 2\Delta_k)^2 - \underline{x}_k^2).$$

Opening parentheses and simplifying the resulting expression, we conclude that $V' - V = (4\Delta_k/n) \cdot (\underline{x}_{k+1} - \underline{x}_k)$. Since V is the maximum, we must have $V' - V \leq 0$, hence $\underline{x}_{k+1} \leq \underline{x}_k$. Due to our ordering, we thus have $\underline{x}_{k+1} = \underline{x}_k$. Since we assumed that $\Delta_k < \Delta_{k+1}$, we have $\overline{x}_k = \underline{x}_k + 2\Delta_k < \overline{x}_{k+1} = \underline{x}_{k+1} + 2\Delta_{k+1}$, hence the interval $\mathbf{x}_k$ is a proper subset of $\mathbf{x}_{k+1}$ – which is impossible.

In the second case, when $\Delta_k \geq \Delta_{k+1}$, let us replace $\overline{x}_k$ with $\overline{x}_k - 2\Delta_{k+1}$ (which is still $\geq \underline{x}_k$), and $\underline{x}_{k+1} = \overline{x}_{k+1} - 2\Delta_{k+1}$ with $\overline{x}_{k+1}$. Here, the average E remains the same, and the only difference between the new value V' of the variance and its old value V comes from the change in terms x_k^2 and x_{k+1}^2, hence

$$V' - V = \frac{1}{n} \cdot (\overline{x}_{k+1}^2 - (\overline{x}_{k+1} - 2\Delta_{k+1})^2) - \frac{1}{n} \cdot (\overline{x}_k^2 - (\overline{x}_k - 2\Delta_{k+1})^2),$$

i.e., $V' - V = (4\Delta_{k+1}/n) \cdot (\overline{x}_{k+1} - \overline{x}_k)$. Since V is the maximum, we must have $V' - V \leq 0$, hence $\overline{x}_{k+1} \leq \overline{x}_k$. Due to our ordering, we thus have $\overline{x}_{k+1} = \overline{x}_k$. Since we assumed that $\Delta_k \geq \Delta_{k+1}$, we have $\underline{x}_k = \overline{x}_k - 2\Delta_k \geq \underline{x}_{k+1} = \overline{x}_{k+1} - 2\Delta_{k+1}$, i.e., $\mathbf{x}_k \subseteq \mathbf{x}_{k+1}$. Since intervals cannot be proper subsets of each other, we thus have $\mathbf{x}_k = \mathbf{x}_{k+1}$. In this case, we can simply swap the values x_k and x_{k+1}, variance will not change.

If necessary, we can perform this swap for all needed k; as a result, we get the maximizing vector with the desired property.

4°. Due to Part 3 of this proof, the desired value $\overline{V} = \max V$ is the largest of $n + 1$ values (2.3) corresponding to $k = 0, 1, \ldots, n$.

In principle, to compute $\overline{V}$, we can therefore compute each of these values and find the largest of them. Computing each value takes $O(n)$ times, so computing $n + 1$ such values would require time $O(n^2)$. Let us show that we can compute $\overline{V}$ faster.

We must find the index k for which V_k is the largest. For the desired k, we have $V_k \geq V_{k-1}$ and $V_k \geq V_{k+1}$. Due to (2.3), we conclude that

$$V_k - V_{k-1} = \frac{1}{n} \cdot \left(\underline{x}_k^2 - \overline{x}_k^2\right)$$

$$- \left(\frac{1}{n} \cdot \sum_{i=1}^{k} \underline{x}_i + \frac{1}{n} \cdot \sum_{i=k+1}^{n} \overline{x}_i\right)^2 + \left(\frac{1}{n} \cdot \sum_{i=1}^{k-1} \underline{x}_i + \frac{1}{n} \cdot \sum_{i=k}^{n} \overline{x}_i\right)^2. \qquad (2.5)$$

Each pair of terms in the right-hand side of (2.5) can be simplified if we use the fact that $a^2 - b^2 = (a - b) \cdot (a + b)$ and use the notations Δ_k and $\widetilde{x}_k \overset{\text{def}}{=} (\underline{x}_k + \overline{x}_k)/2$. First, we get $\underline{x}_k^2 - \overline{x}_k^2 = (\underline{x}_k - \overline{x}_k) \cdot (\underline{x}_k + \overline{x}_k) = -4\Delta_k \cdot \widetilde{x}_k$. Second, we get

$$\left(\frac{1}{n} \cdot \sum_{i=1}^{k-1} \underline{x}_i + \frac{1}{n} \cdot \sum_{i=k}^{n} \overline{x}_i\right)^2 - \left(\frac{1}{n} \cdot \sum_{i=1}^{k} \underline{x}_i + \frac{1}{n} \cdot \sum_{i=k+1}^{n} \overline{x}_i\right)^2 =$$

$$\frac{2}{n} \cdot (\overline{x}_k - \underline{x}_k) \cdot \left(\frac{1}{n} \cdot \sum_{i=1}^{k-1} \underline{x}_i + \frac{1}{n} \cdot \widetilde{x}_k + \frac{1}{n} \cdot \sum_{i=k+1}^{n} \overline{x}_i\right).$$

Here, $\overline{x}_k - \underline{x}_k = 2\Delta_k$, hence the formula (2.5) takes the following form:

$$V_k - V_{k-1} = \frac{4}{n} \cdot \Delta_k \cdot \left(-\widetilde{x}_k + \frac{1}{n} \cdot \sum_{i=1}^{k-1} \underline{x}_i + \frac{1}{n} \cdot \widetilde{x}_k + \frac{1}{n} \cdot \sum_{i=k+1}^{n} \overline{x}_i\right).$$

Since $V_k \geq V_{k-1}$ and $\Delta_k > 0$, we conclude that

$$-\widetilde{x}_k + \frac{1}{n} \cdot \sum_{i=1}^{k-1} \underline{x}_i + \frac{1}{n} \cdot \widetilde{x}_k + \frac{1}{n} \cdot \sum_{i=k+1}^{n} \overline{x}_i \geq 0. \tag{2.6}$$

Substituting the expressions $\underline{x}_i = \widetilde{x}_i - \Delta_i$ and $\overline{x}_i = \widetilde{x}_i + \Delta_i$ into the formula (2.6) and moving all the negative terms to the other side of the inequality, we get the inequality (2.1). Similarly, the inequality $V_{k+1} \leq V_k$ leads to (2.2).

When k increases, the left-hand side of the inequality (2.1) increases – because $\widetilde{x}_k$ increases as the average of the two increasing values $\underline{x}_k$ and $\overline{x}_k$, and the sum is increasing. Similarly, the right-hand side of this inequality decreases with k. Thus, if this inequality holds for k, it should also hold for all smaller values, i.e., for $k-1$, $k-2$, etc.

Similarly, in the second desired inequality (2.2), when k increases, the left-hand side of this inequality increases, while the right-hand side decreases. Thus, if this inequality is true for k, it is also true for $k+1$, $k+2, \ldots$

If both inequalities (2.1) and (2.2) are true for two different values $k < k'$, then they should both be true for all the values intermediate between k and k', i.e., for $k+1, k+2, \ldots, k'-1$. If (2.1) and and (2.2) are both true for k and $k+1$, this means that in both cases, we have equality, thus $V_k = V_{k+1}$, so it does not matter which of these values k we take.

Thus, modulo this equality case, there is, in effect, only one k for which both inequalities are true, and this k can be found by the bisection method as described in the above algorithm.

How long does this algorithm take? In the beginning, we only know that k belongs to the interval $[1, n]$ of width $O(n)$. At each stage of the bisection step, we divide the interval (containing k) in half. After I iterations, we decrease the width of this interval by a factor of 2^I. Thus, to find the exact value of k, we must have I for which $O(n)/2^I = 1$, i.e., we need $I = O(\log(n))$ iterations. On each iteration, we need $O(n)$ steps, so we need a total of $O(n \cdot \log(n))$ steps. With $O(n \cdot \log(n))$ steps for sorting, and $O(n)$ for computing the variance, we get a $O(n \cdot \log(n))$ algorithm. □

3 Second Case: Using a Limited Number of Different Types of Measuring Instruments

In this case, the interval data consists of m families of intervals such that within each family, no two intervals are proper subsets of each other.

Similarly to the proof of Theorem 1, we can conclude that if we sort each family in lexicographic order, then, within each family, the maximum of V is attained on one of the sequences (2.4). Thus, to find the desired maximum $\overline{V}$, it is sufficient to know the value $k_\alpha \le n$ corresponding to each of m families. Overall, there are $\le n^m$ combinations of such values, and for each combination, computing the corresponding value of the variance requires $O(n)$ steps. Thus, overall, we need time $O(n^{m+1})$.

4 Third Case: Privacy in Statistical Databases

When the measurements $\widetilde{x}_i$ correspond to data that we want to keep private, e.g., health parameters of different patients, we do not want statistical programs to have full access to the data – because otherwise, by computing sufficiently many different statistics, we would be able to uniquely reconstruct the actual values $\widetilde{x}_i$. One way to prevent this from happening is to supply the statistical data processing programs not with the exact data, but only with intervals of possible values of this data, intervals corresponding to a fixed partition; see, e.g., [4]. For example, instead of the exact age, we tell the program that a person's age is between 30 and 40.

To implement the above idea, we need to fix a partition, i.e., to fix the values $t_1 < t_2 < \ldots < t_n$. In this case, instead of the actual value of the quantity, we return the partition-related interval $[t_i, t_{i+1}]$ that contains this value.

Privacy-related intervals $[t_i, t_{i+1}]$ satisfy the same property as intervals from the first case: none of them is a proper subset of the other. Thus, we can apply the algorithm described in Section 2 and compute the exact range $\mathbf{V}$ in polynomial time – namely, in time $O(n \cdot \log(n))$.

Acknowledgments

This work was supported in part by NASA grant NCC5-209, by the AFOSR grant F49620-00-1-0365, by NSF grants EAR-0112968, EAR-0225670, and EIA-0321328, by the Army Research Laboratories grant DATM-05-02-C-0046, and by NIH grant 3T34GM008048-20S1.

The authors are thankful to the anonymous referees for the valuable suggestions.

References

1. Cormen Th. H., Leiserson C. E., Rivest R. L., and Stein C.: Introduction to Algorithms, MIT Press, Cambridge, MA, 2001.
2. Ferson, S., Ginzburg, L., Kreinovich, V., Longpré, L., Aviles, M.: Computing Variance for Interval Data is NP-Hard, ACM SIGACT News **33**(2) (2002) 108–118

3. Granvilliers, L., Kreinovich, V., Müller, L.: Novel Approaches to Numerical Software with Result Verification", In: Alt, R., Frommer, A., Kearfott, R. B., Luther, W. (eds.), Numerical software with result verification, Springer Lectures Notes in Computer Science, 2004, Vol. 2991, pp. 274–305.
4. Kreinovich, V., Longpré, L.: Computational complexity and feasibility of data processing and interval computations, with extension to cases when we have partial information about probabilities, In: Brattka, V., Schroeder, M., Weihrauch, K., Zhong, N.: Proc. Conf. on Computability and Complexity in Analysis CCA'2003, Cincinnati, Ohio, USA, August 28–30, 2003, pp. 19–54.

On Efficiency of Tightening Bounds in Interval Global Optimization

Antanas Žilinskas[1,2] and Julius Žilinskas[1]

[1] Institute of Mathematics and Informatics
Akademijos g. 4, MII, 08663, Vilnius, Lithuania
{antanasz,julius.zilinskas}@ktl.mii.lt
[2] Vytautas Magnus University

Abstract. The tightness of bounds is very important factor of efficiency of branch and bound based global optimization algorithms. An experimental model of interval arithmetic with controllable tightness of bounds is proposed to investigate the impact of bounds tightening in interval global optimization. A parallel version of the algorithm is implemented to cope with the computational intensity of the experiment. The experimental results on efficiency of tightening bounds are presented for several test and practical problems. The suitability of mater-slave type parallelization to speed up the experiments is justified.

1 Introduction

Global optimization techniques are generally classified in two main classes: deterministic and stochastic [1,2,3]. An important subclass of deterministic algorithms constitute interval arithmetic-based algorithms [4,5,6]. The advantages of interval global optimization methods are well known. Interval arithmetic-based methods guarantee the prescribed precision of computed solutions. Application of these methods does not suppose any parametric characterizations of objective functions, as for example, a Lipshitz constant is requested by Lipshitz model-based algorithms. There are many examples of global optimization problems successfully solved by interval methods. However, the serious disadvantage of these methods is their computational complexity. Despite of numerous recently proposed improvements, the practical efficiency of interval methods is hardly predictable: some problems are solved rather fast, but some problems of modest dimensionality cannot be solved in acceptable time. Normally the dependency problem is claimed as a main reason of inefficiency since it causes overestimating of function ranges by inclusion functions. In the present paper we investigate how much the overestimation level influences the efficiency of an interval global optimization method. Since the experiments are computing intensive, a parallel implementation of the algorithm is used.

2 Interval Arithmetic Branch and Bound Strategy

A minimization problem $f(x),\ x \in A \subseteq R^n$ is considered assuming that objective function $f(x)$ is continuous, and A is a box, i.e. $A = \left[\underline{a_1}, \overline{a_1}\right] \times \ldots \times \left[\underline{a_n}, \overline{a_n}\right]$. We assume that an interval technique can be applied to calculate upper and lower bounds for values of the function $f(x)$ over a box $B \subset A$; we denote these bounds $\overline{f(B)}$ and $\underline{f(B)}$. Since bounds are available, a branch and bound technique can be applied to construct a global optimization algorithm. Let us denote a current upper bound for the minimum by $\overline{f}$. Only the boxes with $\underline{f(B)} \leq \overline{f}$ can contain a minimizer, and they form a work pool or candidate set for further processing. The boxes with $\underline{f(B)} > \overline{f}$ are deleted. A box is chosen from the candidate set for the subdivision with subsequent updating of the current $\overline{f}$ and of the candidate set. The efficiency of the algorithm depends on choice and on subdivision strategies. However, it depends first of all on early deleting of the non-promising boxes. The latter can crucially depend on the tightness of bounds. In the present paper we experimentally investigate the influence of the tightness of bounds on algorithm efficiency. The quantitative estimates are needed when developing new methods of bound tightening, e.g. by means of stochastic random arithmetic [7,8].

The efficiency of minimization can be enhanced introducing additional deleting strategies. An efficient version of interval global optimization method involving monotonicity and convexity tests and auxiliary interval Newton procedure is proposed in [4]. The method is applicable for twice continuously differentiable objective functions, whose derivatives are available either via mathematical expressions or automatic differentiation. Although these enhancements could be applied for the considered functions they have not been included in our algorithm since they would complicate investigation of the relation between the algorithm efficiency and relative tightness.

The standard choice and subdivision strategies are used in our branch and bound algorithm: a box with minimal $\underline{f(B)}$ is chosen for the subdivision, and the subdivision is implemented as bisection of the longest edges. While these strategies are not necessarily optimal, they provide a basis on which to compare the effects of tightness. The efficiency of the deleting crucially depends on the estimate of the upper bound for the minimum. We have investigated two estimates. The result of both cases are presented below under headings 'interval and real' and 'interval only'. 'Interval and real' means that the smallest of function values at the middle points in the evaluated boxes is used as an upper bound for the minimum. 'Interval only' means that the smallest upper bound over the evaluated boxes is used as an upper bound for the minimum.

We are interested in the number of evaluations of a subroutine calculating bounds as a function of the tightness of bounds. The first extreme case corresponds to the bounds produced by the standard interval arithmetic for computing the inclusion function. The second extreme case corresponds to the exact bounds. The intermediate tightness of bounds are obtained as the convex combination of the extreme cases. The procedure for calculating the exact bounds is an auxiliary procedure needed for our experimental investigation. Of course, such a procedure is not intended for practical implementation since it is too expensive. In our experiments we compute the exact bounds minimizing and maximizing the objective function over the considered boxes. The auxiliary optimization problems are tackled by means of interval global optimization using the same interval arithmetic libraries and including an enhanced strategy of deleting of the non-promising boxes. In this case the optimization algorithm uses interval derivatives and tests for monotonicity and convexity.

The efficiency of tightening bounds is measured using numbers of function evaluations. Optimization time here is not a suitable criterion since large fraction of total time is spent to compute exact bounds, which are used only in experimental investigation. In practical implementations calculation of practically meaningful bounds will, of course, avoid computation of exact bounds.

3 Implementation

The investigation of efficiency of the bounds tightening is computing intensive. Therefore a parallel implementation of the algorithm is needed. We refer to the papers [9,10,11] for the discussion on the performance of parallel implementations of branch and bound algorithms. For our experiments a parallel version of the considered algorithm is constructed using the master-slave paradigm. The master runs the core branch and bound algorithm and holds a candidate set – yet not explored boxes. To model different tightness of bounds the master needs precise bounds for the candidate set boxes. The tasks of estimation of exact bounds are sent to the slaves, who use the bound constrained interval global optimization algorithm including monotonicity and convexity tests. The master-slave paradigm is appropriate here because the slave tasks of global minimization and maximization are time consuming, and therefore communications are not too frequent.

The algorithm is implemented in C++. Two implementations of interval arithmetic are used: the integrated interval arithmetic library [12] for the SUN Forte Compiler and a C++ interval library filib++ [13]. The former implementation is used for parallel and serial version of the algorithm on Sun systems, and the latter is used for its serial version on Linux systems. It has been shown in [14] that these libraries are most fast and accurate. Both libraries support extended interval arithmetic, and use templates to define intervals. Interval definitions in these libraries are similar, and the global optimization algorithm has been adapted so that the used library could be exchanged easily. The algorithm was compiled with SUN Forte and GNU C++ compilers.

A parallel version of the algorithm has been implemented using MPI (Message-Passing Interface – a standard specification for message-passing libraries [15]). The algorithm has been tested and the experiments have been performed on the Sun HPC 6500 UltraSPARC-III based system with Sun MPI.

4 Results of Experimental Investigation

We were interested to estimate quantitatively the influence of bounds' tightening to the number of calls of procedures calculating real and interval objective function values. Test functions for global optimization known from literature and two practical problems were used in our experiments. For the description of the used test functions we refer to [16]. The multidimensional scaling (MDS) problem and the used data are presented in [16] as well. A chemical engineering problem called 'separation problem' is described in [17]. Rosenbrock, Box and Betts, McCormic, Six Hump Camel Back functions were minimized with tolerance 10^{-4}, and the other functions were minimized with tolerance 10^{-2}. The numbers of interval function and real function evaluations NIFE and NFE needed to solve the problems with prescribed tolerance are presented in Table 1. The standard interval arithmetic bounds were used by the implemented optimization algorithm.

Table 1. Experimental estimation of optimization complexity using standard interval arithmetic

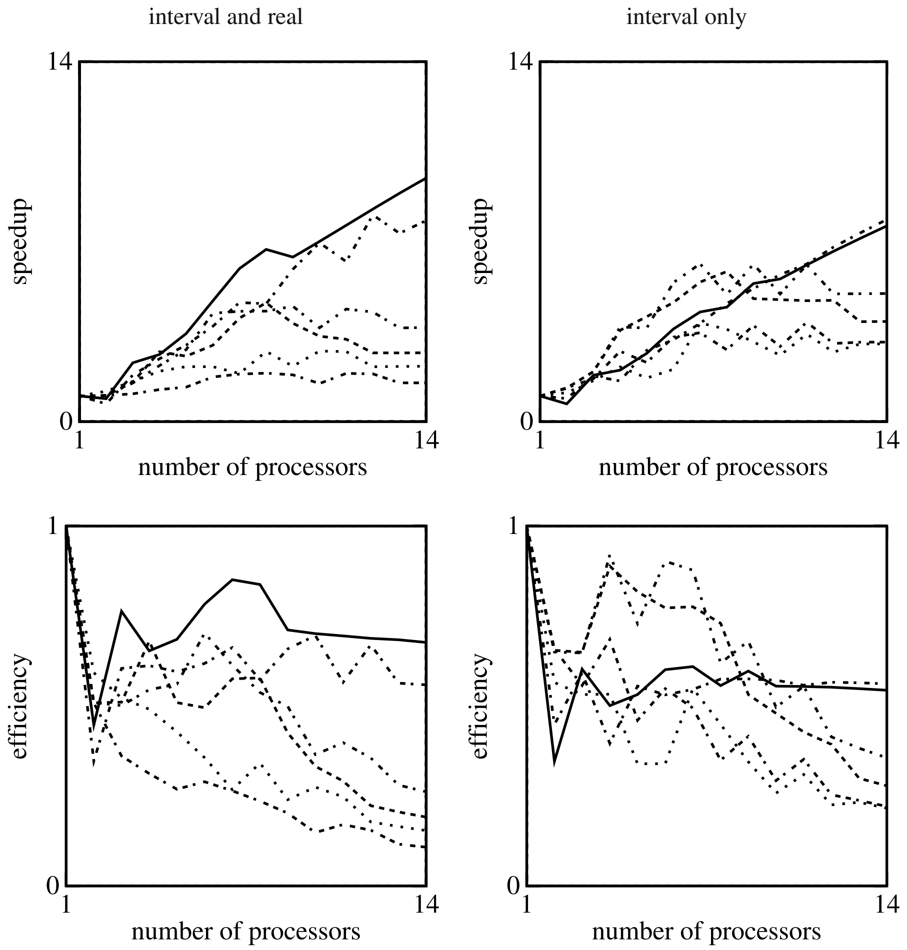
	interval and real		interval only
Test function or problem	NIFE	NFE	NIFE
Rosenbrock	273	141	905
Box and Betts	10367	6502	19029
McCormic	1105657	784452	1568899
Six Hump Camel Back	5935219	4407615	8714819
Goldstein and Price	351219	247359	398117
Multidimensional scaling	66513	47720	119569
Separation problem	102893	70402	363479

Speedup and efficiency criteria was used to evaluate performance of parallel algorithm. The speedup is the ratio of time between the sequential version of algorithm and the parallel one: $s_m = t_1/t_m$, where t_1 is time of sequential algorithm and t_m is time used by the parallel algorithm running on m processors. The efficiency of parallelization is the speedup divided by the number of processors: $e_m = s_m/m$. Speedup and efficiency of parallelization have been estimated for the case of exact bounds, estimated by slaves using enhanced interval minimization and maximization of the objective function over the received box. Up to 14 processors have been used. The estimated criteria are shown in Figure 1. For most functions the highest efficiency of parallelization is achieved with 6-8 processors. For some of test functions (for example, Rosenbrock, Box and Betts, and Multidimensional scaling), efficiency was rather low, since the number of tasks was too small to justify parallelization. However, these functions can be successfully minimized by a sequential algorithm. For other functions improvement is evident, and parallelization significantly speeds up the experiments.

Several versions of the algorithm with 'interval and real' and 'interval only' upper bound for minimum, and several relative tightness of bounds were used. The relative tightness of bounds is specified by a number between 0.0 and 1.0, where 0.0 corresponds to the exact bounds for function values and 1.0 corresponds to the standard interval inclusion. The criteria of the algorithm efficiency are the numbers of bounding function and real function evaluations performed by the algorithm before termination: NBFE and NFE. The results of the experiments are presented graphically to show dependence of NBFE and NFE on relative tightness of bounds.

To evaluate dependency quantitatively we estimate the dependency factor by the mean ratio of the width of the standard inclusion interval to the width of exact bounds for the function values. The dependency factor for each test function and each value of relative tightness shows how much the standard inclusion interval overestimates exact bounds for function values because of the dependency problem.

Results of optimization of the Rosenbrock test function are shown in Figure 2a. The experimental estimates of the dependency factor are 1.0005-1.0010 for the case 'interval



Rosenbrock – dotted line, Box and Betts – dashed line, McCormic – solid line, Six Hump Camel Back – dot dash, Goldstein and Price – dot dot dash, Multidimensional scaling – dot dash dash

Fig. 1. Speedup and efficiency of the parallel algorithms

and real' and 1.0050-1.0053 for the case 'interval only'. These values of dependency factors show that dependency for the Rosenbrock function is very weak. The conclusion that the dependency practically can be ignored explains why the graphs in Figure 2a are practically horizontal. The graphs of NBFE and NFE with respect to relative tightness for the Box and Betts, Six Hump Camel Back, and Goldstein and Price functions are presented in Figures 2b, 2c and 2d correspondingly. The graphs are almost linear. The numbers of objective functions evaluations can be reduced approximately twice by means of improving tightness of bounds. We have not included the graphs for the McCormic test function since qualitatively they seem identical to Figure 2c.

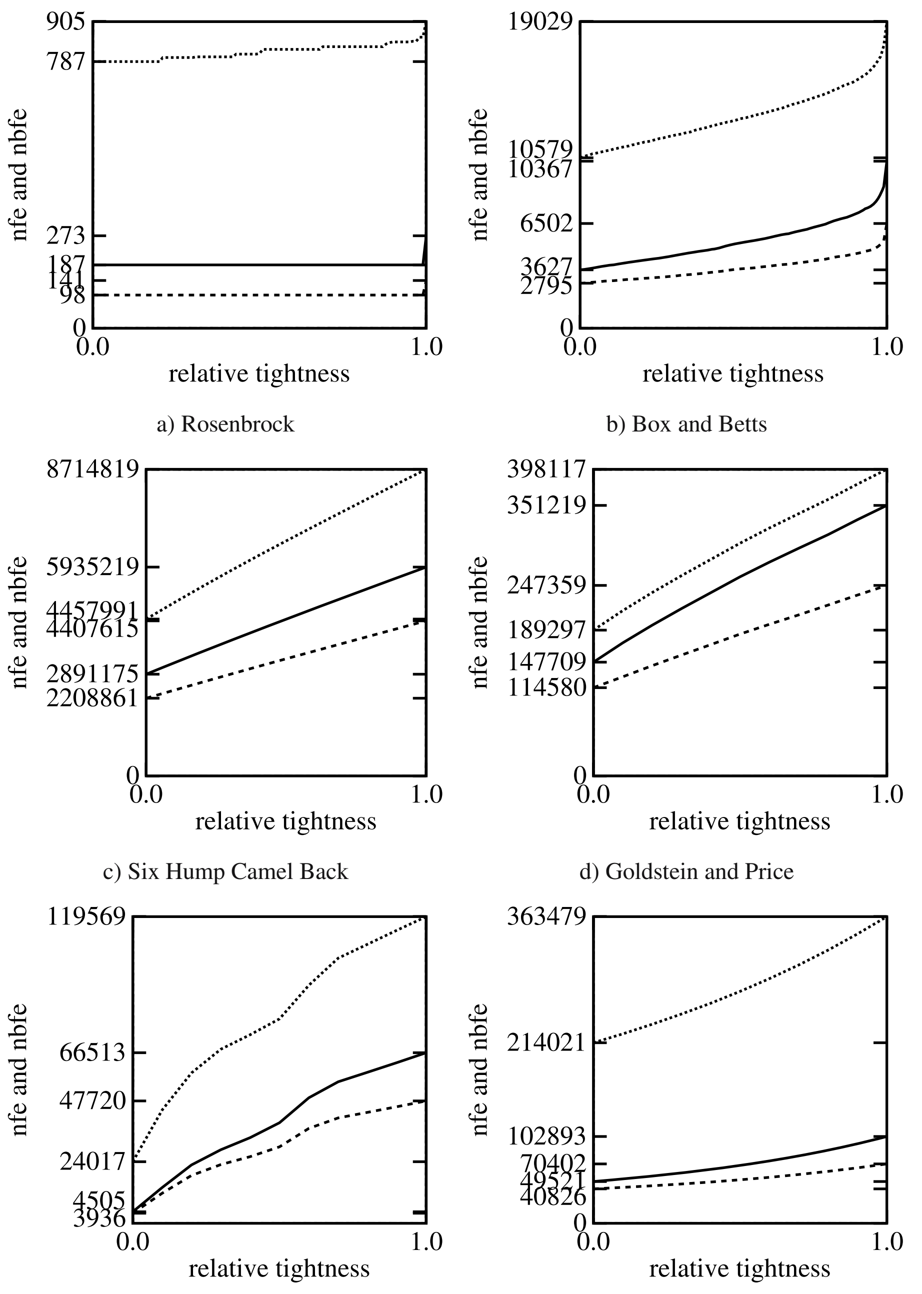


Fig. 2. Efficiency of tightening. Solid line represents numbers of bounding function evaluations and dashed line represents numbers of real function evaluations for the case 'interval and real', dotted line represents numbers of bounding function evaluations for the case 'interval only'

The experimental results concerning two practical problems are presented in Figures 2e and 2f. The impact of dependency factor to the needed number of function evaluations is much larger for MDS problem than for the test functions as well as for the Separation problem.

The experimental estimates of impact of the dependency factor to the optimization complexity are summarized in Table 2, where the first line for each problem corresponds to the case 'interval and real' and the second line corresponds to the case 'interval only'. There is no doubt that improvement of bounds, e.g. by means of elimination of dependency, increases the efficiency of branch and bound global optimization based on interval arithmetic. However, the quantitative estimate of improvement is difficult. From our research we can conclude that for relatively simple test problems the algorithm using exact bounds is at least twice more efficient than the algorithm using standard interval arithmetic inclusion. In special cases, e.g. for MDS problem, such an improvement can be up to 10 times.

Table 2. Experimental estimation of impact of dependency factor to optimization complexity

		interval bounds		exact bounds	
Test function or problem	Dependency factor	NIFE	NFE	NBFE	NFE
Rosenbrock	1.0005–1.0010	98	273	141	187
	1.0050–1.0053	905		787	
Box and Betts	1.1675–1.2362	10367	6502	3627	2795
	1.1544–1.2045	19029		10579	
McCormic	1.6032–1.9829	1105657	784452	466695	394134
	1.6013–1.8497	1568899		788617	
Six Hump Camel Back	1.6497–1.8433	5935219	4407615	2891175	2208861
	1.6489–1.7740	8714819		4457991	
Goldstein and Price	1.6385–1.8726	351219	247359	147709	114580
	1.6344–1.8304	398117		189297	
Multidimensional scaling	1.4623–2.0704	66513	47720	4505	3936
	1.3908–1.5798	119569		24017	
Separation problem	1.1344–1.3080	102893	70402	49521	40826
	1.1154–1.1933	363479		214021	

Although precision of bounds is an important factor of efficiency of a branch and bound algorithm, it does not uniquely define the efficiency. Further investigation is needed taking into account more subtle details of algorithms and objective functions. For example, tightness of bounds is more important where the measure of set $\{x : f(x) < \min_{x \in A} f(x) + \epsilon\}$ is large relatively to A. MDS is namely such a problem. Similar problems are frequent in statistical parameter estimation by least squares.

Sometimes a minimization problem can be solved fast even without tight bounds. This can happen in case when global minimum is much lower than the average value of $f(x)$ over A, and a good estimate $\overline{f}$ is accidentally found. In this case many boxes can be deleted from the candidate set in next iterations after estimation of $\overline{f}$ even using interval arithmetic inclusion.

5 Conclusions

The relative tightness of bounds strongly influences efficiency of global optimization algorithms based on branch and bound approach combined with interval arithmetic. Numbers of objective functions evaluations are approximately linear functions of the relative tightness of bounds. For the standard test functions the branch and bound algorithm using exact bounds is approximately twice more efficient than using the interval arithmetic inclusion. However, in special cases efficiency can be increased up to 10 times. For quantitative estimation of potential improvement of efficiency further investigation is necessary taking into account more subtle details of the algorithm and properties of an objective function.

Parallelization of the algorithm for modeling variable tightness of bounds speeds up the experiments. The master-slave paradigm is suitable for such experiments because the slave tasks of global minimization/maximization are time consuming, and communications are not too frequent.

Acknowledgment

The work of the second co-author is supported by NATO CBP.EAP.RIG.981300 (the Reintegration grant).

The authors thank anonymous reviewers for many constructive remarks enabling us to improve the manuscript.

References

1. R. Horst, and H. Tuy. *Global Optimization: Deterministic Approaches*. 2nd. edn. Springer-Verlag, Berlin Heidelberg New York, 1993.
2. R. Horst, and P. Pardalos. *Handbook of Global Optimization*. Kluwer Academic Publishers, Dodrecht, 1995.
3. A. Törn, and A. Žilinskas. Global optimization. *Lecture Notes in Computer Science*, 350:1–255, 1989.
4. E. Hansen, and G.W. Walster. *Global Optimization Using Interval Analysis*. 2nd. edn. Marcel Dekker, New York, 2003.
5. R.B. Kearfott. *Rigorous Global Search: Continuous Problems*. Kluwer Academic Publishers, Dodrecht, 1996.
6. H. Ratschek, and J. Rokne. *New Computer Methods for Global Optimization*. Ellis Horwood, Chichester, 1995.
7. J. Žilinskas, and I.D.L. Bogle. Balanced random interval arithmetic. *Computers and Chemical Engineering*, 28(5):839–851, 2004.
8. J. Žilinskas, and I.D.L. Bogle. Evaluation ranges of functions using balanced random interval arithmetic. *Informatica*, 14(3):403–416, 2003.

9. J. Clausen. Parallel branch and bound – principles and personal experiences. In Migdalas, A., Pardalos, P.M., Storøy, S., eds.: *Parallel Computing in Optimization*. Kluwer Academic Publishers, 239–267, 1997.
10. C.Y. Gau, and M.A. Stadtherr. Parallel branch-and-bound for chemical engineering applications: Load balancing and scheduling issues. *Lecture Notes in Computer Science*, 1981:273–300, 2001.
11. B. Gendron, and T.G. Crainic. Parallel branch-and-bound algorithms – survey and synthesis. *Operations Research*, 42(6):1042–1066, 1994.
12. SUN Microsystems. *C++ Interval Arithmetic Programming Reference*. Forte Developer 6 update 2 (Sun WorkShop 6 update 2), 2001.
13. M. Lerch, G. Tischler, J.W. von Gudenberg, W. Hofschuster, and W. Krämer. The interval library filib++ 2.0 - design, features and sample programs. Preprint 2001/4, Universität Wuppertal, 2001.
14. J. Žilinskas. Comparison of packages for interval arithmetic. *Informatica*, 16(1):145–154, 2005.
15. Message Passing Interface Forum. *MPI: A message-passing interface standard (version 1.1)*. Technical report, 1995.
16. K. Madsen, and J. Žilinskas. *Testing branch-and-bound methods for global optimization*. Technical report 05/2000, Technical University of Denmark, 2000.
17. T. Csendes. Optimization methods for process network synthesis – a case study. In Carlsson, C., Eriksson, I., eds.: *Global & multiple criteria optimization and information systems quality*. Abo Academy, Turku, 113–132, 1998.

Trends in Large Scale Computing: An Introduction

Organizer: Scott B. Baden

Department of Computer Science and Engineering
University of California, San Diego, USA
`http://www-cse.ucsd.edu/~baden`

Software infrastructure for large scale computation often fails to realize the full potential afforded by technological advances, and the result is lost opportunities for making scientific discovery. This workshop will examine two important issues in software infrastructure for large scale computation: achieving scalability, and optimization through specialization. Three presentations address scalability: tolerating latency, and aggregating and analyzing voluminous simulation and performance data. Two presentations address optimization: run time optimization of task graphs and compile time techniques for incorporating semantic information into the optimization of object oriented frameworks.

Speakers:

- Scott B. Baden, Department of Computer Science and Engineering, University of California, San Diego, USA
- Susanne M. Balle, Hewlett-Packard Company, Nashua, NH, USA
- Olav Beckmann, Department of Computing, Imperial College, London, UK
- Dan Quinlan, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory
- Allan Sussman, Department of Computer Science and UMIACS, University of Maryland, Chaos Project Faculty, USA

Ygdrasil: Aggregator Network Toolkit for Large Scale Systems and the Grid

Susanne M. Balle, John Bishop, David LaFrance-Linden, and Howard Rifkin

Hewlett-Packard
110 Spit Brook Road, Nashua, NH, 03062, USA
{Susanne.Balle,John.Bishop,David.LaFrance-Linden,
Howard.Rifkin}@hp.com

Abstract. Many tools such as user programming tools, performance tools and system administration tools, which are targeted to be used on a very large number of processors (1,000+) have the same common problems. They need to deal with the large amount of data created by the large number of processes as well as be scalable. The data need to be condensed and reorganized into a useful set of information before it is presented to the user. We present an aggregating network toolkit, which due to its modularity can be tailored to almost any purpose involving reduction of data, from a large number of individual back-ends (connected to a tool or a user application), to a user portal. The latter allows the user to visualize as well as re-arrange the data to fit his need (integration with statistical packages, visualization, etc.). The user can write his own plug-ins and thereby tell the network what is getting aggregated, how to aggregate the data (identical data, discard data, etc.), as well as how to display the data in the user portal.

1 Introduction

With the expansion of the grid, more and more users (developers as well as system administrators) have to deal with larger and larger systems. Users want to run their application on larger and larger number of processors as their application becomes grid-enabled [1,2]. System administrators want to manage thousands of processors in an easy way without having to go through tedious operations. Along with the large number of processes, they have to deal with heterogeneous systems composed of differing operating systems and hardware. Both types of users need tools. Developers need user-programming tools such as debuggers, performance monitoring tools, etc. Other users need tools for system administration tasks, database retrieval, etc.

The common problems for all these tools are (1) dealing with the huge amount of output the thousands or tens of thousands processes generate, (2) present these outputs to the user in an understandable way, (3) launch efficiently a job or a tool onto a set of nodes given a specified topology and (4) scalability. In the case of (1), the data are often identical (example debugger, booting process, cluster-wise shell).

We present enhancements to the aggregator network infrastructure described in Balle [3] as well as make it available to users as a standalone toolkit. The main idea behind the toolkit is to give the tool writers as well as advanced users the opportunity to take advantage of the aggregating network infrastructure with little work from their part.

2 Prior Work

The aggregating network approach described in Balle [3] and related work by Roth [4] present approaches which try to solve problems (1) to (4). Analytic work regarding the reduction of output is presented in Waheed [5].

The aggregating network described in [3] is composed of a tree-like network and at each node in the tree, aggregators are inserted. The tree-like network reduces the message latency to $O(log(n))$ instead of $O(n)$ in traditional design. Along with decreasing the message latency, this approach introduces parallelism into the overall architecture. Each message from the back-ends (see Fig. 1) can flow down to the next level in the tree without being delayed or stalled due to synchronization with other components in the network. This design allows the aggregators and back-ends to work independently (therefore in parallel). The aggregators condense the output of the tools into a smaller and more manageable sub-set, which is then displayed to the user. Each aggregator condenses the output it receives from the back-end under its control independent of its peers at the same level in the tree. The architecture's parallelism is limited by how fast the slowest tool responds to a given event or by a time-out mechanism [3]. The aggregators are distributed onto the available nodes using a load-balanced scheme.

The aggregators condense outputs from the tools/aggregators at each level in the tree. In order to handle the huge amount of output from the tools (in [3] the tool is a debugger), outputs are categorized and aggregated using the following guidelines:

- Type 1: Identical outputs from each of the tools/aggregators,
- Type 2: Identical outputs apart from containing different numbers and
- Type 3: Widely differing outputs.

It is easy to extend this categorization as well as these aggregation guidelines to include binary outputs and additional aggregating strategies.

Using the aggregating tree network approach, described in Balle [3], creates a powerful, scalable and user-friendly parallel debugger. Performance results show that the debugger scales well both in the startup phase and with respect to the user commands's response time. A user only has to wait 31 seconds for the debugger to startup up on 2700 processors [3].

The Ladebug MPP project [3] illustrates how to achieve a parallel, scalable debugger based on using a serial command line debugger on each node. During that project we came to realize that the infrastructure could be used to create other types of tools such as system administrators tools, that monitor the status of systems in a cluster, cluster booting tools, parallel versions of performance tools such as Paradyn [18], PAPI [19], etc. The Ygdrasil toolkit allows tool writers to create all kind of tools. Only people's imagination, not technology, should be the limit to what they want to do.

The Ygdrasilt toolkit provides a number of enhancements to the work described in Balle [3]:

- A tool independent portable infrastructure: to allow tools writers to develop parallel versions of their favorite serial tools as well as of their own tools.
- Customizable toolkit via plug-ins: to provide the ability to customize the components to do specify tasks, the rest of the features in the toolkit are easily enhanceable as well. For example it is trivial to add support for a new job launcher.

- User portal (UP)/GUI: to allow users to customize the displayed output. Users can connect UPs to already existing Application Portals (AP) and check on the status of a session. This is useful if we had used Ygdrasil as a launcher and wanted to check on the status of a running job at a later time. Another usage could be if we know that our job fails after running for 3 or 4 hours under debugger control, this feature allows us to connect back via a home PC or a handheld to an existing session and continue the debugging session. The same is true for any session using a tool created with the Ygdrasil toolkit.
- Minimal Grid support

Each of these features are discussed in details in Sections 3, 4 and 5.

3 Ygdrasil Toolkit

The Ygdrasil toolkit is composed of an aggregator tree-like network, a parallel tool and network startup mechanism, and is customizable via components plug-ins.

The main idea behind the toolkit is to give the tool writers as well as advanced users the opportunity to take advantage of the aggregating network infrastructure and create their own tools with very little work from their part. The toolkit provides all the necessary “plumbing" to create scalable tools. Tool writers only have to concentrate their effort on what is really of interest to them namely their tool.

The toolkit can be used to create tools for large scale systems as well as in very limited cases in a Grid environment. Enhancing the toolkit for a full Grid environment is the topic of future research.

The Ygdrasil toolkit is composed of four major components interconnected with an aggregator network infrastructure as illustrated in Fig. 1. The four specialized components are: the User Portal (UP), the Application Portal (AP), the Aggregator Node (AN), and the Backend (BE). A plug-in, which is part of each component, allows users to customize each component of the toolkit. An example of such customization is described in Section 5. The toolkit is very modular which makes it very flexible and thereby useful for a large range of purposes.

Each component has a specific role within the toolkit. The UP handles all user interactions and passes user commands to the AP. Within the network (from UP to BE and back), the user commands and other messages, which are sent up and down the tree, are encapsulated as objects. Information passing is done via serialization of specialized objects [7]. Each component identifies the type of object it receives, unpacks the data (if needed), do what it needs to do for that specific object type given the specific plug-in, repackages (if needed) the resulting data into an object which is then sent to the next component. The AP, a persistent component, integrates with launchers such as globus_run [2], mpirun (mpich) [8], etc. to retrieve the needed job information to launch the tools. The user can launch the application directly under the control of a tool such as a debugger if needed. The AP controls the other components in the tree and can be considered as the root of the tree. The AP passes data/objects back and forth in the tree as well as massaging the data according to the AP plug-in. Going leaf-ward (from the UP to the BE), the AN distributes commands. Going root-ward (from the BE to the UP), the AN reduces data. The plug-in allows users to tell each component what they want it

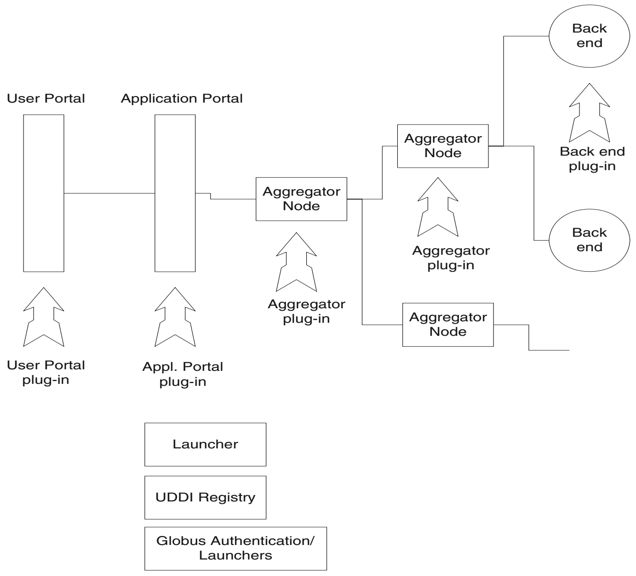


Fig. 1. Ygdrasil Infrastructure

to do. In Section 5, we describe a debugger plug-in where the plug-in in the AN reduces data/objects by aggregating similar outputs. The BE is the leaf node of the tree and is the interface between the tool and the Ygdrasil network.

The tool writer or advanced user can customize the network to suit his needs via the plug-in toolkit. The latter is composed of all the individual plug-ins (UP, AP, AN, BE). As part of our prototyping phase, we wrote two plug-in toolkits: (1) a debugger plug-in which works with serial command line debuggers such as GNU gdb, Compaq Ladebug [9], and HP's wdb and (2) a memory checking tool plug-in. The debugger plug-in is very similar in functionality to the Compaq Ladebug parallel debugger [9] and is described in Section 5.

The Ygdrasil toolkit does not directly interact with the parallel application. Instead it interacts with a user specified tool, e.g. debugger, that will run on the compute nodes. An exception would be if advanced users develop their application using the Ygdrasil toolkit as their middleware. The encapsulation of the data into known objects needs to be specified by the users since tools can be externally supplied. As mentioned earlier, each major component is designed to use a user specified plug-in which implements that component's function for a specific tool.

The Ygdrasil toolkit is written in Java [7] for portability. We use Java serialization over TCP to communicate between components. In the future we will integrate with the Globus communication fabric [2]. We then expect users to be able to select their preferred communication protocol.

An example of how we expect users, tools and the Ygdrasil toolkit to interact in a Grid environment is illustrated in Fig. 2. Fig. 2 shows a user application running across multiple clusters. We assume that each of the clusters is located in geographically

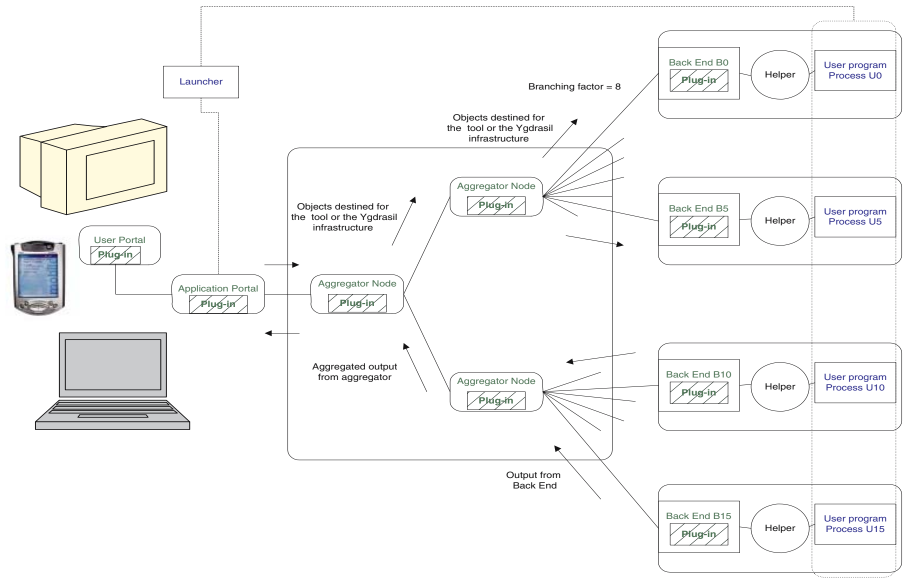


Fig. 2. Aggregator tree-like network

different locations. A user launches his grid-enabled application [1,2] and then wants to start a tool (debugger, performance monitoring tool, etc.) on the remote nodes. The user can, via the user portal, launch an aggregating tree-like network infrastructure which will interconnect each of the individual tools. Each tool is connected to a back-end node. The tool can be a stand-alone tool (shell, performance tool, etc.) or attached to the user application (debugger, message passing tool, etc.). The user portal can either be started up at the user's desktop or the user can connect back into an already existing job. A special interface for handhelds such as the IPaQ [6] is provided so that the user can check the status of his job as well as get a summary of certain properties either default or pre-selected properties.

The tree-like network is composed of aggregator nodes which can be placed either on the compute clusters or on middleware servers. It is easy to imagine that in the future these middleware servers could be located at miscellaneous ISPs between the end-user and the compute nodes.

4 User Portal

The UP displays the "customized" outputs from the tools connected to the Ygdrasil Aggregator Network and allows users to interact with the network. The UP can be considered to be a client to the AP (see Fig. 1).

The UP's tool panel is customized via a plug-in as illustrated in Fig. 1. An example of such a customization is described in Section 5. The current version of the UP has been tested on Tru64 [10], HP-UX [11], Windows ME/2000/XP [12], Alpha Linux [13], and on handhelds Windows CE/PocketPC [14], Familiar Linux [15], and in a browser.

5 Debugger Plug-In

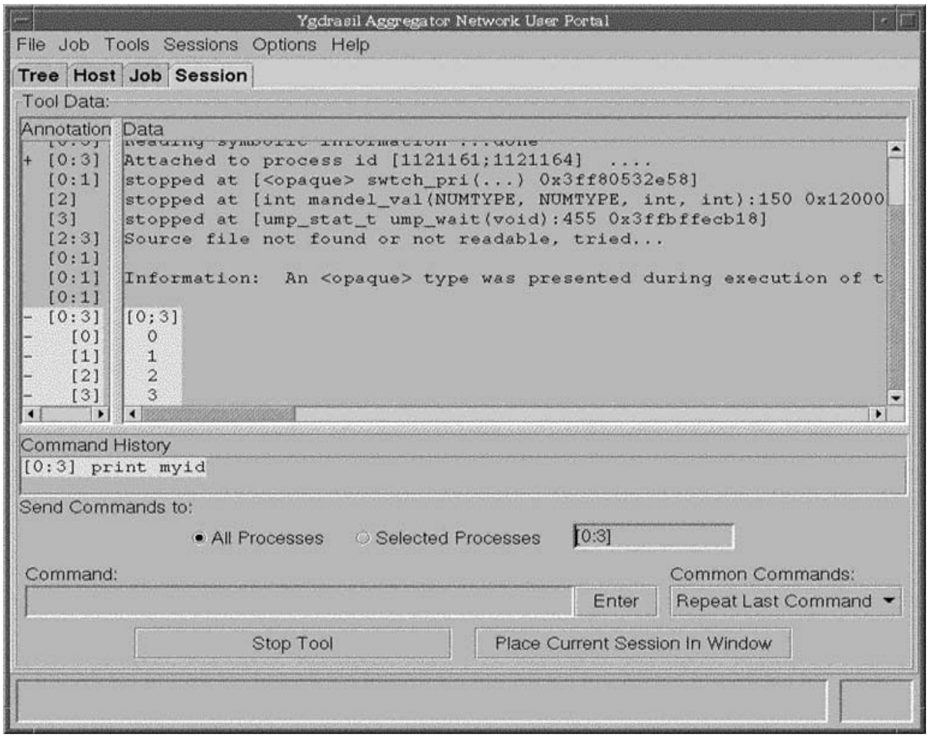
In this Section we briefly describe how we have customized the toolkit and thereby its UP, AP, AN, and BE plug-ins to create a scalable debugger with features and capabilities similar to the scalable debugger described in [3]. The main difference with [3] is that we get the same look-and-feel for 3 different command line debuggers.

We have created plug-ins for the GNU gdb debugger, the Compaq Ladebug debugger [5] and HP's gdb variant wdb which allow users to transparently use the same tool with either of the debuggers. The look-and-feel is the same independent of what debugger is used. Users can either select a specific debugger or let the network choose based on the platform-OS combination the debugger is used on. As mentioned earlier the BE is the leaf node of the tree and is the interface between the tool and network. The role of the debugger BE Plug-in is to translate output from the debuggers into something that the aggregators and the network can understand. In the case of the debugger, the BE plug-in has to do the initial bookkeeping such as tagging individual output, tagging outputs that belongs together, etc. as well as package the output into tagged objects that can be transmitted root-ward (from BE to UP). The debugger BE Plug-in also has a predefined set of default parameters such as preferred debugger for a specific platform-OS combination, etc.

The AN is a complex component which plays a role in the leaf-ward direction (from UP to BE) as well as in the root-ward direction. The AN captures the unique tag and other relevant information of objects that are sent leaf-ward for later use when correlating output objects will be coming root-wards. In the root-ward direction, it unpacks the objects, redirects the unpacked object to the correct plug-in, hosts the necessary timeout mechanism to avoid severe bottlenecks in the flow of objects, repackages objects from the plug-in into a tagged object which is sent to the next level in the tree. The debugger AN plug-in ensure that objects received from the BE or ANs are aggregated correctly and uses sophisticated timeout mechanisms to aggregate outputs in an optimal way even when network performance or tool response time aren't at their best.

The AP component and its debugger plug-in are mainly pass through components. The UP component and its plug-in are shown in Fig. 3. The window is divided into two panels: an annotation panel and a data panel. In the annotation panel is listed the processor set corresponding to the output listed in the data panel. As mentioned in Section 2, we have aggregated the data into Type 1, Type 2 and Type 3 objects. In the case of Type 2 aggregation (identical outputs apart from containing different numbers), we have placed a "+" in the annotation panel. The user can click on the "+" and the objects will expand and let the user see all the individual outputs. An example of Type 2 is shown below: $+[0:3][0:3]$ which expands to

Annotation:	Data:
- [0:3]	[0:3]
- 0	0
- 1	1
- 2	2
- 3	3



```
+ [0:3]   Attached to process id [1121161;1121164]  ....
  [0:1]   stopped at [<opaque> swtch_pri(...) 0x3ff80532e58]
  [2]     stopped at [int mandel_val(NUMTYPE, NUMTYPE, int, int):150 0x12000
  [3]     stopped at [ump_stat_t ump_wait(void):455 0x3ffbffecb18]
  [2:3]   Source file not found or not readable, tried...
  [0:1]
  [0:1]   Information:  An <opaque> type was presented during execution of t
  [0:1]
- [0:3]   [0;3]
-    [0]    0
-    [1]    1
-    [2]    2
-    [3]    3
```

```
[0:3] print myid
```

```
0:3]
```

Fig. 3. User Portal with Debugger Plug-in

When the user clicks on the “-" in front of the Type 2 output, the displayed object retracts itself and only the Type 2 output is shown: $+ [0:3][0:3]$.

The Ygdrasil toolkit and the debugger plug-ins have been tested on Tru64 [10], Linux (Alpha) [13], and HPUX (PA-Risc and ia64) [11].

6 Experimental Results

The timings reported in this section were achieved on the Pittsburgh Supercomputing Center’s Alphaserver SC cluster [16]. The Sierra Cluster supercomputer [16,17] is composed of 750 Compaq Alphaserver ES45 nodes and a separate front-end node. Each computational node contains four 1-GHz processors, totaling 3000 processors, and runs the Tru64 Unix operating system. A Quadrics interconnection network connects the nodes [16]. We chose to time the Ygdrasil toolkit startup when using the Ladebug debugger Plug-ins since the serial Ladebug debugger is available on the Terascale Computing System AlphaServer SC cluster at Pittsburgh Supercomputing Center.

6.1 Startup

The parallelism in the startup is achieved by having each aggregator at a given level startup the aggregators or leaf debuggers in its control at the next level. Each level

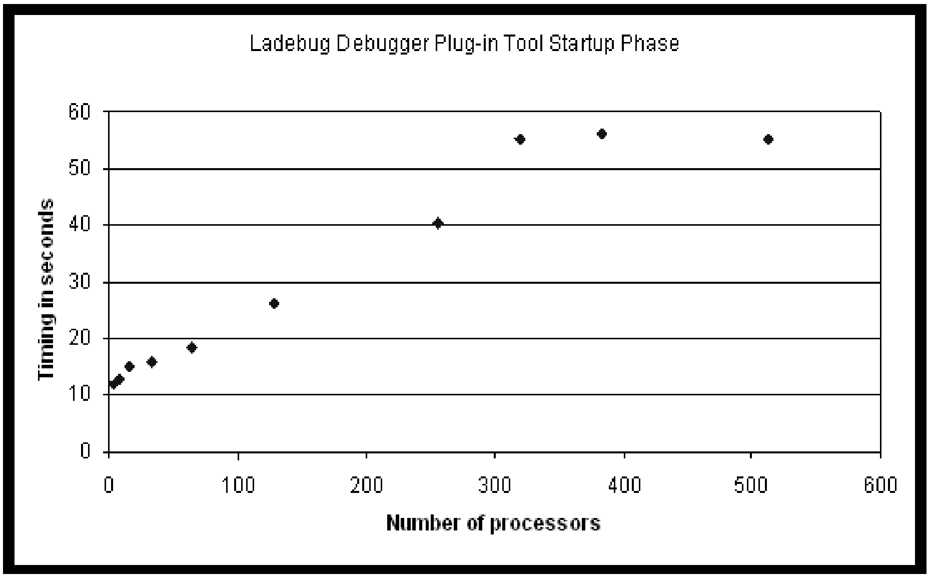


Fig. 4. Ladebug Debugger Plug-in Toolkit startup

can therefore be started in parallel. The amount of parallelism in the startup is mainly limited by the interactions between the debugger and the operating system. There is an initial serialization to gather the process information and calculate the specifics of the network tree. The approach implemented in the Ygdrasil toolkit is similar to the approach described in Balle [3].

Fig. 4 presents the Ygdrasil toolkit using the Ladebug debugger plug-ins' startup time as a function of the number of processors n. We chose a branching factor of 8 which means that the tree will have two levels of aggregators for $n = 64$, three levels for $n = 512$, etc.

When $n \leq 320$, the startup time increases by 137 milliseconds/processor. The time increase per processor is due to the fact that each process has to read the hostname table before it can determine where to startup its children processes. The reading of the hostname information is a serial process and will therefore prohibit the startup mechanism from being fully processed in parallel. This limitation is known and will be part of future improvements to the toolkit. When $n > 320$ we notice that the startup time is close to constant. We do expect that the startup time will increase again as the number of processors and the number of aggregator level increases. Unfortunately we only had limited access to the cluster and were not able to run on more than 512 processors.

7 Conclusion

The Ygdrasil toolkit is a modular and flexible infra-structure. We have shown that the startup phase is scalable up to 500 processors and expect it to scale to the thousands of processors. The flexibility of the toolkit was one of our major goals while designing and implementing the toolkit. We didn't want to limit the toolkit's users' imaginations. We envision that the toolkit can be used to create tools i.e. our debugger plug-in, standalone application i.e. genealogy data mining, etc.

The toolkit has achieved its goals namely - give the tool writers as well as advanced users the opportunity to create scalable tools or applications by taking advantage of the aggregating network infrastructure with little work from their part. They can concentrate on what really interest them: their tool. The toolkit solves the common problem of collecting, organizing, and presenting, in a scalable manner, the output of large scale computations.

At SC2002, we presented a debugger prototype based on the Ygdrasil toolkit which allowed users to debug an application using the Compaq ladebug debugger on Compaq Tru64 platforms and the wdb debugger (HP's variant of gdb) on HP HPUX platforms from the same UP.

In order to maximize the usability of the Ygdrasil toolkit we are currently investigation a possibility of open sourcing the toolkit.

Acknowledgements

The authors wish to thank J. Ray Scott and Sergiu Sanielevici from the Pittsburgh Supercomputing Center for letting us get access to the Terascale Computing System AlphaServer SC cluster. Finally, we thank Rae Eastland for his Grid-service contribution, Jim Bovay for supporting the project, and the anonymous referees for suggesting improvements to the manuscript.

References

1. I. T. Foster and C. Kessleman. The Globus Project: a status report. *Seventh Heterogeneous Computing Workshop*, Orlando, Florida, March 1998.
2. Globus Toolkit 2.0 (www.globus.org).
3. S. M. Balle, B. R. Brett, C. P. Chen, and D. LaFrance-Linden. Extending a Traditional Debugger to Debug Massively Parallel Applications. *Journal of Parallel and Distributed Computing*, Volume 64, Issue 5 , May 2004, Pages 617-628.
4. P. C. Roth, D. C. Arnold, and B. P. Miller. MRNet: A Software-Based Multicast/Reduction Network For Scalable Tools. *SC 2003*, Phoenix, AZ, November 2003.
5. A. Waheed, D. T. Rover, and J. K. Hollingsworth. Modeling and Evaluating Design Alternatives for an On-Line Instrumentation System: A Case Study. *IEEE Transactions on Software Engineering*, Vol 24, No 6, June 1998.
6. http://www.handhelds.org/geeklog/index.php.
7. Java 2 Platform. Version 1.3, Standard Edition (J2SE). http://java.sun.com/
8. MPICH. http://www-unix.mcs.anl.gov/mpi/mpich/
9. Ladebug Debugger Manual Version 67. Compaq Computer Corporation, February 2002. http://www.compaq.com/ladebug.
10. Tru64 UNIX for HP AlphaServer Systems. http://h30097.www3.hp.com/
11. HPUX UNIX. http://www.hp.com/products1/unix/operating/
12. Windows 2000. http://www.microsoft.com/windows2000/
13. HP Alpha Linux. http://h18002.www1.hp.com/alphaserver/linux/

14. Window CE/Pocket PC. (http://www.microsoft.com/windowsmobile/products/pocketpc/)
15. Familiar Linux. http://familiar.handhelds.org/
16. Compaq Computer Corporation. Alphaserver SC: Scalable Supercomputing, July 2000. Document number 135D-0900A-USEN.
17. Pittsburgh Supercomputing Center. http://www.psc.edu
18. Paradyn http://www.cs.wisc.edu/ paradyn/
19. PAPI http://icl.cs.utk.edu/papi/

Enabling Coupled Scientific Simulations on the Grid

Alan Sussman and Henrique Andrade

Department of Computer Science
University of Maryland
College Park, Maryland
{als,hcma}@cs.umd.edu

Abstract. This paper addresses the problem of providing software support for simulating complex physical systems that require multiple physical models, potentially at multiple scales and resolutions and implemented using different programming languages and distinct parallel programming paradigms. The individual models must be coupled to allow them to exchange information either at boundaries where the models align in physical space or in areas where they overlap in space. Employing multiple physical models presents several difficult challenges, both in modeling the physics correctly and in efficiently coupling multiple simulation codes for a complete physical system. As a solution we describe InterComm, a framework that addresses three main parts of the problem: (1) providing comprehensive support for specifying at runtime what data is to be transferred between models, (2) flexibly specifying and efficiently determining when the data should be moved, and (3) effectively deploying multiple simulation codes in a high performance distributed computing environment (the Grid).

1 Introduction

Simulation of physical systems has become the third main form of investigation in many scientific disciplines, along with theory and experimentation. There have been many software projects and tools that have addressed issues related to efficiently supporting running simulations of individual physical models of such systems on large scale parallel and distributed systems. However, not much attention has been paid to software support for simulation of complex physical systems requiring multiple physical models. One reason is that there have not been many efforts in scientific disciplines to model complex phenomena using multiple models. This is now changing in areas such as earth science, space science, and other physical sciences, where the use of multiple physical models can provide great advantages over single models.

Employing multiple physical models presents several difficult challenges, both in modeling the physics correctly, and in efficiently using multiple simulation codes to model a complete physical system. In particular, the individual models must be *coupled* to allow them to exchange information either at boundaries where the models align in physical space, or in areas where the models overlap in space. Providing effective software support for building and deploying coupled simulations will have an immediate impact in multiple scientific disciplines that have realized the necessity for coupling models at multiple scales and resolutions. Such support will also have a wide impact in the future as comprehensive modeling of additional complex physical systems becomes more common.

1.1 Space Weather Modeling – An Example of Coupled Simulations

An interesting simulation coupling example comes from the domain of space science, in particular space weather modeling. With collaborators at the Center for Integrated Space Weather Modeling at Boston University, we have been working on a set of simulation codes that model both the large scale and microscale structures and dynamics of the Sun-Earth system as depicted in Figure 1. The models include the SAIC corona code [14], the Maryland-Dartmouth solar wind code [15], the LFM global magnetohydrodynamics (MHD) magnetospheric code [5], the TING ionosphere-thermosphere code [20], and the microscale Hall MHD, hybrid, and particle codes [18]. Each code is among the best regarded for modeling its respective niche, and they span the physics regimes of the Sun-Earth system covering many, if not all, of the physical regimes needed for a comprehensive model. They also present most of the coupling issues that will be faced in coupling other physical systems. While ad-hoc coupling works, it provides no generality for further simulation extensions; each new case must be done from scratch. What these efforts show, however, are the issues that a general infrastructure must address. While these models cover a multitude of different spatial scales and physical models, they share one feature. All use spatial grids to organize the computations, in this case structured grids, as do most space physics codes. The issue of dealing with grids, both structured and unstructured, is central to the design of InterComm.

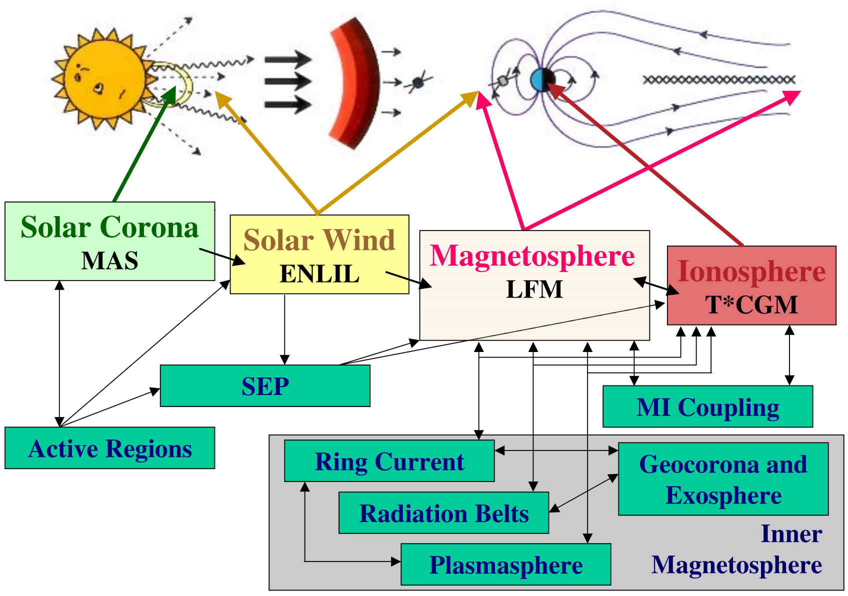


Fig. 1. The coupled space physics models

In Section 2, we present the InterComm framework, show how it addresses the question of code interoperability and describe how employing the InterComm framework facilitates the deployment of coupled codes on the Grid. In Section 3 we discuss how

InterComm compares to existing computational science tools. Finally, in Section 4, we conclude with a discussion of the set of InterComm features already available and present the set of design and development activities that we are currently pursuing.

2 Overview of the InterComm Framework

The InterComm framework addresses the problem of providing support for coupled simulations by allowing for direct model to model data transfers. Three issues are addressed: (1) Providing comprehensive support for specifying at runtime *what* data is to be moved between simulations, (2) Flexibly specifying and efficiently determining *when* the data should be moved, and (3) Effectively deploying multiple coupled simulation codes in a high performance distributed computing environment (i.e., the Computational Grid [6]). For all parts of the problem, a major goal is to minimize the changes that must be made to each individual simulation code, because those codes are generally very complex and difficult to modify and maintain.

The Data Coupler Infrastructure

The foremost issue to be addressed when coupling two physical simulation codes consists of enabling the exchange of data between them. In InterComm, support is provided for many different kinds of parallel (and sequential) programs, written in multiple languages (including C, C++, Fortran77, Fortran90, Java, etc.) and using common scientific parallel programming techniques such as message-passing and multi-threading.

The elementary entity required for enabling communication to/from a parallel program is the **data descriptor**. A data descriptor essentially defines the distribution of a data structure across multiple address spaces (i.e., processes or processors). For an array data structure, the descriptor contains information about where each array element is located, namely which process owns the element and where in the address space of the owner process the element is located. The InterComm framework provides an API for building the descriptors necessary to describe the data distributions utilized by all the individual models and for specifying the part of the data structure to move. The initial array distributions we support include regular and generalized block distributions, as seen in Figure 2, and completely irregular, or explicit, distributions [9].

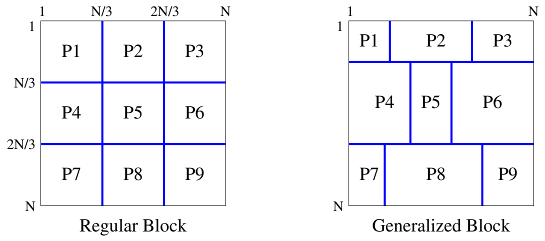


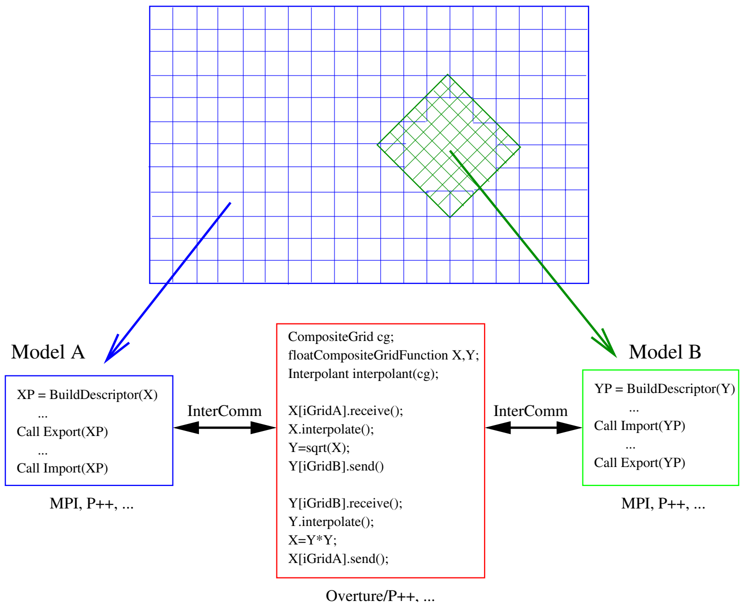
Fig. 2. Block data distributions for an $N \times N$ two-dimensional array

Conceptually, only three communication calls are required for exchanging data: (1) *Export* makes a consistent version of the distributed data structure available for a potential data transfer; (2) *Import* requests that a consistent version of a matching data structure be copied into the argument data structure (the semantics of the matching process are described later in this section); and (3) *BuildDescriptor* builds the data descriptor required so that the framework can schedule and perform the needed interprocessor communication across all processes in each program. If one side of the data transfer is a parallel program, all three calls on that side are collective across the processes in that program, meaning that the call occurs in all processes, with appropriately matching parameters (i.e., the data descriptor and subset specification must match, but the actual data will likely reside in different memory locations in different processes). Given data descriptors for both sides of the data transfer, and specifications of the subsets of the data structures that will participate in the transfer, the InterComm framework can build a *communication schedule* (i.e., a description of the set of messages that must be sent between each pair of processes) for a matching export/import pair. Schedules can be built and reused, since they are explicit objects in the model codes.

In the near future, InterComm will also provide support for translation of data between models. From the point of view of the computational scientist, a very important feature of the framework is that it allows existing codes to be coupled in most cases by modifying no more than a few lines of code in the existing models. There may be a substantial amount of work (and code) required to translate or interpolate from one physical model to another (or the grids used in the models). However, that work can all be done externally to the existing codes and only a few communication calls need to be inserted in the original programs. Figure 3 shows a conceptual picture of how two model codes, A and B, will be coupled, including the data translation component.

The Control Infrastructure

A second problem in supporting the exchange of data between two models consists of mechanisms for describing *when* data will be exported/imported. Specifying when a data structure (or some part of) should be transferred between model components can be solved either statically or dynamically. A dynamic coordination language approach [8] could be employed, providing arbitrary expressive power, but we are concerned with being able to efficiently decide when a data transfer should take place. For this reason InterComm employs a static approach, whereby a *mapping specification* is provided when the various model components begin execution. For each pair of matched imported/exported data structures in different components, the mapping specification describes which *export* call should be used to satisfy a given *import* call (e.g., export calls may not result in data transfers if no other program imports that data). The mapping specification design relies on the notion of a *timestamp*, which must be provided as a parameter to every export and import call in a model component. A timestamp is a floating point number (or a simple counter) intended to capture the time-dependent nature of the solution to most models of physical systems, and can often be directly taken or derived from the time step used in the simulation code. For any mapping specification, one guarantee that must be made to be able to determine whether exported data is needed (to avoid storing it indefinitely) is that timestamps from a single model are monotonically non-decreasing.



```
XP = BuildDescriptor(X)
    ...
Call Export(XP)
    ...
Call Import(XP)
```

```
CompositeGrid cg;
floatCompositeGridFunction X,Y;
Interpolant interpolant(cg);

X[iGridA].receive();
X.interpolate();
Y=sqrt(X);
Y[iGridB].send()

Y[iGridB].receive();
Y.interpolate();
X=Y*Y;
X[iGridA].send();
```

```
YP = BuildDescriptor(Y)
    ...
Call Import(YP)
    ...
Call Export(YP)
```

Fig. 3. A simple example illustrating how the framework can be used to couple two Fortran codes. **B** is a computation embedded in the larger simulation **A**. There is an overlap region for the two computations where information needs to be passed back and forth. Array variable **X** in **A** provides the necessary information to **B** in the overlap region, but **B** requires **Y** for its computation. **A**, in turn, requires an update on **X** in its overlap region with **B**. The center of the diagram shows the inter-grid and data translation program

This guarantee is enforced by InterComm, implying that once an import statement in a program requests a data structure with a timestamp with value i, the program will never request one with a value smaller than i.

A Grid and Component-Based Framework

Another facet of the coupled simulation problem lies in effectively deploying the coupled simulation models in the high performance distributed computing environment. The individual model codes have different computational and storage requirements and may be designed to run in different sequential and parallel environments. Deploying coupled simulations on the Grid requires starting each of the models on the desired Grid resources, then connecting the models together via the InterComm framework.

One difficult issue is the initial *discovery* problem. InterComm will make use of existing Grid discovery service mechanisms (e.g., the Globus Monitoring and Discovery Service, MDS [3]), to allow the individual components to register, to locate appropriate resources for each component, and also to enable components to find each other. In practice, adhering to Grid services interfaces means that InterComm will deploy its services through Web and Grid service compliant interfaces.

3 Related Work

While there have been several efforts to model and manage parallel data structures and provide support for coupled simulation, they all have significant limitations. Parallel programming tools such as Overture/P++ [1] and MPI [19] provide support for developing *single* applications programs on high-performance machines. Parallel Application Work Space (PAWS) [11] and the Model Coupling Toolkit (MCT) [12] provide the ability to share data structures between parallel applications. MCT has been developed for the Earth System Modeling Framework (ESMF) [4]. PAWS and MCT determine inter-program communication patterns within a separate component, while in InterComm such patterns are generated directly in the processes that are part of the parallel applications. Collaborative User Migration, User Library for Visualization and Steering (CUMULVS) [7] is a middleware library that facilitates the remote visualization and steering of parallel applications and supports sharing parallel data structures between programs. Roccom [10] is an object-oriented software framework targeted at parallel rocket simulation, but does not address high performance inter-component communication. Our prior work on the data coupler part of the overall problem led to the design and development of the Meta-Chaos library [17], which makes it possible to integrate data parallel components (potentially written using different parallel programming paradigms) within a single application. A complication in using MetaChaos is that it requires that *inquiry functions* be produced for each data distribution. Producing those inquiry functions can be a difficult task and InterComm addresses this issue by obtaining the data distribution information from each application component. On the other hand, an effort that we intend to leverage is being carried out by the Common Component Architecture (CCA) Forum [2]. CCA comprises a set of common interfaces to provide interoperability among high performance application components. In particular, the CCA MxN working group has designed interfaces to transfer data elements between parallel components, and InterComm can be used to implement that interface.

4 Conclusions and Future Work

Currently the data movement middleware for InterComm is fully functional, containing support for multiple languages and parallel programming paradigms [13] (e.g., C, C++, P++/Overture, Fortran77, and Fortran90), and the control part is in an advanced stage of development. The initial coupling efforts are now underway, employing some of the space weather modeling codes. In addition, we are addressing several ongoing research issues: (1) The initial description of the portions of the data to be moved is explicit. For example, for a two-dimensional array data structure, two pairs of array indices are required to describe a contiguous subarray. However, the spatio-temporal characteristics of most physical models can enable specifying the data to be moved by a multi-dimensional bounding box. We plan to investigate using such techniques to allow implicit specification of the part(s) of the data structure that will participate in the transfer; (2) The data descriptors described in Section 2 can also be useful to enable model components to interact with other software tools, such as debuggers (including performance debugging tools such as Paradyn [16]), visualization tools, checkpointing tools, etc. We plan to investigate how these applications can further benefit from the information stored in data descriptors; and, finally, (3) Our team is actively involved in the CCA MxN design and development effort [2].

References

1. D. L. Brown, G. S. Chesshire, W. D. Henshaw, and D. J. Quinlan. Overture: An object oriented software system for solving partial differential equations in serial and parallel environments. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, 1997.
2. Common Component Architecture Forum. *http://www.cca-forum.org*.
3. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC 2001)*. IEEE Computer Society Press, Aug. 2001.
4. Earth System Modeling Framework (ESMF), 2003. *http://www.esmf.ucar.edu/*.
5. J. Fedder, S. Slinker, J. Lyon, and R. Elphinstone. Global numerical simulation of the growth phase and the expansion onset for substorm observed by Viking. *Journal of Geophysical Research*, 100:19083, 1995.
6. I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure (Second Edition)*. Morgan Kaufman / Elsevier, 2003.
7. G. A. Geist, J. A. Kohl, and P. M. Papadopoulos. CUMULVS: Providing Fault-Tolerance, Visualization and Steering of Parallel Applications. *International Journal of High Performance Computing Applications*, 11(3):224–236, Aug. 1997.
8. D. Gelernter and N. Carriero. Coordination languages and their significance. *Commun. ACM*, 32(2), Feb. 1992.
9. High Performance Fortran Forum. HPF language specification, version 2.0. Available from *http://www.netlib.org/hpf*, Jan. 1997.
10. X. Jiao, M. Campbell, and M. Heath. Roccom: An object-oriented, data-centric software integration framework for multiphysics simulations. In *Proceedings of the 2003 International Conference on Supercomputing*, pages 358–368. ACM Press, June 2003.
11. K. Keahey, P. Fasel, and S. Mniszewski. PAWS: Collective Interactions and Data Transfers. In *Proceedings of the Tenth IEEE International Symposium on High Performance Distributed Computing (HPDC 2001)*. IEEE Computer Society Press, Aug. 2001.
12. J. W. Larson, R. Jacob, I. Foster, and J. Guo. The Model Coupling Toolkit. In *Proceedings of International Conference on Computational Science*, 2001.
13. J. Y. Lee and A. Sussman. Efficient communication between parallel programs with Inter-Comm. Technical Report CS-TR-4557 and UMIACS-TR-2004-04, University of Maryland, Department of Computer Science and UMIACS, Jan. 2004.
14. J. Linker, Z. Mikic, D. Biesecker, R. Forsyth, S. Gibson, A. Lazarus, A. Lecinski, P. Riley, A. Szabo, and B. Thompson. Magnetohydrodynamic modeling of the solar corona during whole sun month. *Journal of Geophysical Research*, 104:9809–9830, 1999.
15. R. McNutt, J. Lyon, C. Goodrich, and M. Wiltberger. 3D MHD simulations of the heliosphere-VLISM interaction. In *AIP Conference Proceedings 471: Solar Wind Nine*, page 823. American Institute of Physics, 1999.
16. B. P. Miller, M. D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K. L. Karavanic, K. Kunchithapadam, and T. Newhall. The Paradyn parallel performance measurement tool. *IEEE Computer*, 28(11):37–46, Nov. 1995.

17. J. Saltz, A. Sussman, S. Graham, J. Demmel, S. Baden, and J. Dongarra. The high-performance computing continuum: Programming tools and environments. *Commun. ACM*, 41(11):64–73, Nov. 1998.
18. M. Shay. *The Dynamics of Collisionless Magnetic Reconnection*. PhD thesis, University of Maryland, College Park, 1999.
19. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra. *MPI–The Complete Reference, Second Edition*. Scientific and Engineering Computation Series. MIT Press, 1998.
20. W. Wang, T. Killeen, A. Burns, and R. Roble. A high-resolution, three dimensional, time dependent, nested grid model of the coupled thermosphere-ionosphere. *Journal of Atmospheric and Terrestrial Physics*, 61:385–397, 1999.

High Performance Linear Algebra Algorithms: An Introduction

Organizers: Fred G. Gustavson[1] and Jerzy Waśniewski[2]

[1] IBM T.J. Watson Research Center
Yorktown Heights NY 10598, USA
fg2@us.ibm.com
[2] Department of Informatics & Mathematical Modeling
of the Technical University of Denmark
DK-2800 Lyngby, Denmark
jw@imm.dtu.dk

1 Introduction

This Mini-Symposium consisted of two back to back sessions, each consisting of five presentations, held on the afternoon of Monday, June 21, 2004. A major theme of both sessions was novel data structures for the matrices of dense linear algebra, DLA. Talks one to four of session one all centered on new data layouts of matrices. Cholesky factorization was considered in the first three talks and a contrast was made between a square block hybrid format, a recursive packed format and the two standard data structures of DLA, full and packed format. In both talks one and two, the first two new formats led to level three high performance implementations of Cholesky factorization while using exactly the same amount of storage that standard packed format required. Of course, full format requires twice the amount of storage of the other three formats. In talk one, John Reid presented a definitive study of Cholesky factorization using a standard block based iterative Cholesky factorization, [1]. This factorization is typical of Lapack type factorizations; the major difference of [1] is the type of data structure it uses: talk one uses square blocks of order `NB` to represent a lower (upper) triangular matrix. In talk two, Jerzy Waśniewski presented the recursive packed format and its related Cholesky factorization algorithm, [2]. This novel format gave especially good Cholesky performance for very large matrices. In talk three, Jerzy Waśniewski demonstrated a detailed tuning strategy for talk one and presented performance results on six important platforms, Alpha, IBM, Intel, Itanium, SGI and Sun. The performance runs covered the algorithms of talks one and two as well as Lapack's full and packed Cholesky codes, [3]. Overall, the square block hybrid method was best but was not a clear winner. The recursive method suffered because it did not combine blocking with recursion, [4]. Talk four, presented by Fred Gustavson, had a different flavor. Another novel data format was described which showed that two standard full format arrays could represent a triangular or symmetric matrix using only a total storage that was equal to the storage of standard packed storage, [5]. Therefore, this format has the same desirable property of standard full format arrays: one can use standard level 3 BLAS, [6] as well as some 125 or so full format Lapack symmetric / triangular routines on it. Thus new codes written for the new format are trivial to produce as they mostly consist of just calls to already existing codes. The last talk of session one, by James Sexton was on the massively parallel IBM BG/L machine consisting of up to 65,536 processors. Jim presented some early Linpack benchmark numbers which were quite impressive, [7]. A surprising property of this machine was its low power requirement which was only about 10 % of the current state-of-art massively parallel designs.

In the second session five talks were given on varied topics. Talk one, presented by Carstin Scholtes considered a direct mapped cache design and it then showed how source codes for sparse dense matrix multiply and sparse Cholesky factorization could be examined to derive a set of calculations that could be evaluated in constant time to yield a probabilistic determination of the number of cache misses to be expected when the program was executed on a specific architecture, [8]. The second talk, presented by Fred Gustavson, described a new algorithm for the level 3 BLAS DGEMM on a computer with $M + 1$ levels of memory, M caches and main memory. Its main contribution was to extend the basic two level algorithm $M = 1$ in two ways: (1) streaming was added to the basic two level algorithm and (2) the basic two level algorithm was shown to work at level $i + 1$ to level i for $i = 1, \ldots, M$. These two ideas were combined with another idea: conservation of matrix data. The combination of these three ideas allowed one to show that one could reduce the number of basic DGEMM algorithms from 6^{M+1} to just 4, [9]. Paper three, presented by Keshav Pingali was about a general purpose compiler that obtains high performance for some common dense linear algebra codes, [10]. This method was a deterministic method for finding key hardware parameters of an architecture as opposed to the AEOS approach of the ATLAS project, [11]. Paper four was on software testing of Library Software and in particular, the Lapack Library, [12]. It was presented by Tim Hopkins. This paper defines a quantitative measure of software quality (metrics) and a way to reduce the amount of software testing by allowing tests that fail to improve the metrics to be discarded. The paper observes how Lapack's testing codes actually “measure up" in a software engineering sense. Their initial results indicate that many of Lapack's test codes fail to improve the coverage of their metrics and that many sections of Lapack's code are never executed by the testing software. Some of this lack of coverage is easily remedied; other parts require expertise about the codes being tested. The last paper by Peter Drackenberg is somewhat related to the theme of new data structures of DLA. His approach is to describe a data type for dense matrices whose primitive operations are decomposition and composition (of submatrices) as opposed to indexed element access which is the primitive operation on conventional arrays. This work appears to be related to the Lapack approach and recent work on recursion and iteration done by many authors. Performance results on SGI Octane Systems for level 3 BLAS were found to be encouraging.

Not all of these talks appears in these Lecture Notes. Only talk four of Session one, [5] and talks two, four and five of Session two, [9,12,13] appear. However, for completeness the following references [1,2] covers talks one to three of Session one. An excellent set of slides is available for talk five of Session one and talk one of Session two, [7,8]. [10] covers talk three of Session 2.

References

1. B. S. Andersen, J. A. Gunnels, F. G. Gustavson, J. K. Reid and J. Waśniewski. A Fully Portable High Performance Minimal Storage Hybrid Cholesky Algorithm. *ACM TOMS*, Vol. 31, No. 2 June 2005, pp. 201-227.
2. B. S. Andersen, F. G. Gustavson, and J. Waśniewski. A Recursive Formulation of Cholesky Factorization of a Matrix in Packed Storage. *ACM TOMS*, Vol. 27, No. 2 June 2001, pp. 214-244.
3. E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, D. Sorensen: LAPACK Users' Guide 3rd Ed., Society for Industrial and Applied Mathematics, pub.
4. F. G. Gustavson and I. Jonsson. Minimal Storage High Performance Cholesky via Blocking and Recursion *IBM Journal of Research and Development*, Vol. 44, No. 6, Nov. 2000, pp. 823,849.
5. J. A. Gunnels, F. G. Gustavson. A New Array Format for Symmetric and Triangular Matrices. *Para'04 Workshop on State-of-the-Art in Scientific Computing*, J. J. Dongarra, K. Madsen, J. Waśniewski, eds., Lecture Notes in Computer Science (this proceedings). Springer-Verlag, 2004.
6. J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1-17, Mar. 1990.
7. The BlueGene/L Supercomputer Architecture. J. C. Sexton. Slides on Para04 website
8. C. Scholtes. A Method to Derive the Cache Performance of Irregular Applications on Machines with Direct Mapped Caches. Slides on Para04 website
9. J. A. Gunnels, F. G. Gustavson, G. M. Henry, R. A. van de Geijn. A Family of High-Performance Matrix Multiplication Algorithms. *Para'04 Workshop on State-of-the-Art in Scientific Computing*, J. J. Dongarra, K. Madsen, J. Waśniewski, eds., Lecture Notes in Computer Science, (this proceedings). Springer-Verlag, 2004.
10. J. A. Gunnels, F. G. Gustavson, K. Pingali, K. Yotov. A General Purpose Compiler that obtains High performance for Some Common Dense Linear Algebra Codes. Slides on Para04 website
11. R. Clint Whaley, Antoine Petitet, Jack J. Dongarra. Automated Empirical Optimization of Software and the ATLAS Project. *LAWN Report #147*, Sept., 2000, pp. 1-33.
12. D. J. Barnes, T. R. Hopkins. Applying Software Testing Metrics to Lapack. *Para'04 Workshop on State-of-the-Art in Scientific Computing*, J. J. Dongarra, K. Madsen, J. Waśniewski, eds., Lecture Notes in Computer Science, (this proceedings). Springer-Verlag, 2004.
13. N. P. Drackenberg. A Matrix-type for Performance Portability. *Para'04 Workshop on State-of-the-Art in Scientific Computing*, J. J. Dongarra, K. Madsen, J. Waśniewski, eds., Lecture Notes in Computer Science, (this proceedings). Springer-Verlag, 2004.

Applying Software Testing Metrics to Lapack

David J. Barnes and Tim R. Hopkins

Computing Laboratory, University of Kent
Canterbury, Kent, CT2 7NF, UK
{d.j.barnes,t.r.hopkins}@kent.ac.uk

Abstract. We look at how the application of software testing metrics affects the way in which we view the testing of the Lapack suite of software. We discuss how we may generate a test suite that is easily extensible and provides a high degree of confidence that the package has been well tested.

1 Introduction

Good software engineering practice decrees that testing be an integral activity in the design, implementation and maintenance phases of the software life cycle. Software that executes successfully on an extensive, well constructed suite of test cases provides increased confidence in the code and allows changes to and maintenance of the code to be closely monitored for unexpected side effects. An important requirement is that the test suite evolves with the software; data sets that highlight the need for changes to be made to the source code should automatically be added to the suite and new tests generated to cover newly added features.

To gauge the quality of a test suite we require quantitative measures of how well it performs. Such metrics are useful to developers in a number of ways; first, they determine when we have achieved a well-defined, minimal level of testing; second, they can reduce the amount (and, thus, the cost) of testing by allowing tests that fail to improve the metric to be discarded and, third, they can provide a starting point for a search for new test cases.

In this paper we discuss the strategy that has been used for testing the Lapack suite of linear algebra routines and we quantify the level of testing achieved using three test metrics. Lapack is unusual amongst numerical software packages in that it has been updated and extended over a substantial period of time and several public releases have been made available in that time. It is thus possible to obtain a change history of the code and thus identify maintenance fixes that have been applied to the software either to remove errors or to improve the numerical qualities of the implementation.

In section 2 we discuss the testing metrics we are using and why we would consider using more than one of these measurements. In section 3 we describe the testing strategies used by the Lapack package and discuss how the use of the proposed testing metrics alters the way in which we test some parts of the software.

Section 4 presents our results and our conclusions are given in section 5.

2 Software Testing Metrics

Although some work has been done on the application of formal methods to provide the basis for correctness proofs of linear algebra software and formal derivation of source code (see, for example, Gunnels [5]) the vast majority of numerical software in day-to-day use has been produced by more traditional styles of software development techniques that require extensive testing to increase confidence that it is functioning correctly.

One of the most potent and often underused tools available to assist with this process is the compiler. Most modern compilers will perform a variety of static checks on the source code for adherence to the current language standard (for example, ANSI Fortran [9] and ANSI C [8]). Some (for example, Nag [14] and Lahey [11]) go further and provide feedback on such things as undeclared variables, variables declared but never used, variables set but never used, etc. Although these often only require cosmetic changes to the software in some cases these messages are flagging genuine errors where, for example, variable names have been misspelt.

Furthermore, with judicious use of compilation options, compilers will now produce executable code that performs extensive run-time checks that would be exceedingly difficult to perform in any other way. Almost all will check that array indexes are within their declared bounds but many will go further and check, for example, that variables and array elements have been assigned values before they are used. Such checks are useful even where correctness proofs are available, since simple typographical errors are common wherever human editing is involved in the implementation or maintenance processes.

While it is true that such checking imposes a run-time overhead that would make the executable code produced unusable for many applications where high-performance is essential, it should be mandatory that these capabilities are used during the development, testing and maintenance stages of any software project.

Thus our first requirement is that all the software compiles without any warning messages and that no run-time errors are reported when running the test suite and all the possible execution-time checks have been enabled for the compilation system being used. For such a requirement to have teeth, we must have confidence that the test suite thoroughly exercises the source code.

In order to determine the effectiveness of the testing phase, quantitative measurements are required. Most of the basic testing metrics are white-box and aim to measure how well the test data exercises the code. The simplest is to calculate the percentage of basic blocks (linear sections of code) that are executed. While 100% basic-block coverage is considered by many experts [10] to be the weakest of all coverage criteria its achievement should always be regarded as an initial goal by testers. After all how can we have confidence that software will perform as required if blocks of statements are never executed during the testing process?

Confidence in the software can be further enhanced by extending the testing process to provide more stringent forms of code coverage, for example, branch coverage and linear code sequences and jumps (LCSAJ).

For branch coverage we attempt to generate data sets that will ensure that all branches within the code are executed. For example, in a simple **if-then-endif** block we seek data that will cause the condition to be both true and false. We note here that 100% branch coverage implies 100% basic-block coverage. Thus in the example above basic-block coverage would only require the test to be true in order to execute the statements within the **if**; branch coverage testing would require data to be found that causes the test to be false as well.

The LCSAJ metric measures coverage of combinations of linear sequences of basic blocks, on the grounds that computational effects in a single basic block are likely to have impacts on the execution behavior of immediately following basic blocks. Errors not evident from the analysis of a single basic block may be revealed when executed in combination with other basic blocks, in the same way that module integration testing often reveals further errors beyond those revealed by individual module testing. However, since some calculated LCSAJ combinations may be infeasible, for instance if they involve contradictory condition values, a requirement of 100% LCSAJ coverage is unreasonable and a more modest figure such as 60% would be more approachable.

For the testing metrics described above the amount and quality of test data required to obtain a determined percentage of coverage will generally grow as we progress from basic-block coverage, through branch coverage to LCSAJ testing. Even more thorough testing procedures are available, for example, mutation testing [15] and MC/DC [6]. We do not consider these in this paper.

To be able to obtain metrics data requires tool support. Some compilers provide the ability to profile code execution (for example Sun Fortran 95 Version 7.1 [16] and the NagWare tool *nag_profile* [13]) although, typically, this only provides basic-block coverage.

Tools which perform more sophisticated profiling tend to be commercial; we are using the LDRA Fortran Testbed [12] to generate basic-block and branch coverage and LCSAJ metrics.

3 Testing Lapack

The Lapack suite [1] consists of well over a thousand routines which solve a wide range of numerical linear algebra problems. In the work presented here we have restricted our attention to the single precision real routines thus reducing the total amount of code to be inspected to a little over 25% of the total.

The first release of Lapack was in 1992. Part of the intention of the package's designers was to provide a high quality and highly-portable library of numerical routines. In this, they have been very successful. Since its early releases, which were written in Fortran 77, the software has been modernized with the release of Lapack 95 [2]. Lapack 95 took advantage of features of Fortran 95, such as optional parameters and generic interfaces, to provide wrapper routines to the original Fortran 77 code. This exploited the fact that parameter lists of some existing routines were effectively configured by the types of their actual parameters, and that calls to similar routines with different precisions could be configured using this type information. These wrappers provided the end user with improved and more robust calling sequences whilst preserving the investment in the underlying, older code. Finally, while the injudicious use of wrapper routines may cause performance overheads, this is unlikely to be a problem for the high level user callable routines in Lapack.

Comprehensive test suites are available for both the Fortran 77 and Fortran 95 versions of the package and these have provided an excellent starting point for the focus of our work on the testing of the package. Indeed a core component of the Lapack package is its testing material. This has been an important resource in assisting with the implementation of the library on a wide variety of platform/compiler combinations. However, little, if any, measurement appears to have been made of how well the testing material actually performs from a software engineering sense.

Finally, a package of this size and longevity is potentially a valuable source of data for the evaluation of software quality metrics. Over the past 12 years there have been eight releases of Lapack and this history has allowed us to track the changes made to the software since its initial release. This has provided us with, among other things, data on the number of faults corrected and this has enabled us to investigate whether software quality metrics can be used to predict which routines are liable to require future maintenance (see [3] for further details). It has also been possible to determine whether the associated test suites have been influenced by the fault history.

The main purpose of the supplied testing material is to support the porting process. An expectation of this process is that all of the supplied tests should pass without failures. Following on from the work by Hopkins [7], we were keen to explore how safe this expectation was, by using state-of-the-art compile-time and run-time checks. We began by executing all the supplied test software using a variety of compilers (including NagWare Fortran 95, Lahey Fortran 95 and Sun Fortran 95) that allowed a large number of internal consistency checks to be performed. This process detected a number of faults in both the testing code and the numerical routines of the current release, including accessing array elements outside of their declared bounds, type mismatches between actual arguments and their definitions and the use of variables before they have been assigned values. Such faults are all non-standard Fortran and could affect the final computed results. In all, some 50 of the single precision real and complex routines were affected.

The test-rigs provided are monolithic in that each generates a large number of datasets and routine calls. The number and size of the datasets used are controlled by a user supplied configuration file with the majority of the data being generated randomly and error exits tested separately (but still inside the monolithic drivers and outside of user control). We used the default configuration file in our analysis. Results obtained from each call are checked automatically via an oracle rather than against predefined expected results. This approach has a number of disadvantages

1. there is no quantitative feedback as to how thorough the actual testing process is,
2. running multiple datasets in a single run masks problems that could be detected easily (for example, via run-time checking) if they were run one at a time,
3. the use of a large number of randomly generated datasets is very inefficient in that, from the tester's point of view, the vast majority are not contributing to improving any of the test metrics.

Having removed all the run-time problems mentioned above our next goal was to reduce the number of tests being run (i.e., the number of calls being made to Lapack routines) by ensuring that each test made a positive contribution to the metrics being used. In order to preserve the effort already invested in testing the Lapack routines it was decided to extract as much useful data as possible from the distributed package. To do this we modified *nag_profile* [13] so that it generated basic-block coverage data (profiles) for each individual call to an Lapack routine. We discovered that, for most of the routines, we could obtain the same basic-block coverage using only a small fraction of the generated test sets; see Section 4 for details. Scripts were constructed to extract relevant datasets and to generate driver programs for testing each routine. In order to keep these driver programs simple we only extracted legal data, preferring to run the error exit tests separately; this ensured that such tests were generated for all the Lapack routines.

The routines within the Lapack package split into two distinct subsets; those that are expected to be called directly by the user of the package and those that are only envisioned as being called internally and not described in the user manual. The testing strategy currently employed by the Lapack 77 test suite actually goes further and creates first and second class user callable routines. For example, the testing of the routine SGBBRD is piggy-backed onto the data used in the testing of a more general routine. This leads to 13 out of the 111 basic blocks in SGBBRD being untested; of these 11 were concerned with argument checking for which no out-of-bounds data were provided as it was not treated as a first class citizen — such data had already been filtered out by its caller. The other two were in the main body of code. The data used by the test program was generated from a standard data file that defines the total bandwidth. This routine also requires the number of sub- and super- diagonals to be specified and this pair of values is generated from the total bandwidth value. Unfortunately the method fails to produce one of the special cases.

However there is a deeper testing problem revealed by this example. When testing we usually generate data which constitutes a well-defined and well documented, high level problem (for example, a system of linear equations), we call the relevant solver and then check the computed results. We assume that it is relatively straightforward to perform checking at this level of problem definition. But, having run enough data sets to obtain 100% basic-block coverage of the user callable routine we often find that one of the internally called routines has a much lower coverage. The problem now for the tester is *can they actually find high level user data that will provide the required coverage in the internal routines?* In many cases the answer will be *no*. For example, when using many of the level 1 Blas routines the only possible stride value may be one; no data to the higher level routines will then test the non-unit stride sections of the Blas code.

Where it is impossible for user callable routines to fully exercise internal routines, there are two possibilities. Either require full coverage to be demonstrated through unit tests that are independent of higher level calls, or supplement the higher level calls with additional unit test data for the lower level calls. The former approach is less fragile than the latter, in that it retains complete coverage even in the face of changes to the data sets providing coverage of the higher level routines. On the other hand, it conflicts with the goal of trying to minimise duplicate and redundant coverage.

Over the course of its history, Lapack has undergone a number of revisions, which have included the usual modifications typical of a package of this size: addition of new routines, bug fixes, enhancements of routines, minor renaming, commentary changes, and so on. Somewhat surprising is that the test coverage does not always appear to have been extended to reflect these changes. For instance between version 2.0 and 3.0, several changes were made to SBDSQR, such as the section concerned with QR iteration with zero shift, but many of these changes remain untested by the suite, as they were before the modification.

4 Results

Out of the 316 single precision routines (s*.f) executed when running the Lapack 77 testing software 14,139 basic blocks out of a total of 16,279 were executed. This represents a basic-block coverage metric of 86.9% which is extremely high for numerical software; many packages tested achieve values closer to 50%. Of the approximately 2,000 unexecuted blocks many are due to the lack of coverage of argument checking code. However, there are still a substantial number of statements concerned with core functionality that require data sets to be provided at a higher problem definition level.

Ideally we would like each test to count; each test should cause basic blocks to be executed that no other test exercises. To obtain some idea of the possible degree of redundancy of the supplied tests we looked at individual execution profiles. We define a profile as a bit string that shows the basic blocks executed by each call to a routine; i.e., the string contains a one if the block has been executed and zero if it has not. Note that we cannot extract any path information from this.

The Lapack 77 test cases cause almost 24 million subroutine calls to be made (note that this does not include low level routines like LSAME which was called over 150 million times.) The most called routine was SLARFG which accounted for over 1.7 million calls. This routine illustrates the difficulty in obtaining complete coverage as a side effect of calling higher level routines in that out of the 12 basic blocks that make up this routine one still remained unexecuted.

More telling was that out of more than 20 million profiles obtained less than ten thousand were distinct. Note that it is not the case that we can reduce the number of subroutine calls down to 10,000 since repeated profiles will occur in higher level routines to generate different profiles within lower level routines and vice versa. However it is clear that we should be able to reduce the total number of tests being performed substantially without reducing basic-block coverage. In SLARFG only 3 distinct profiles were generated.

SSBGVD provides a good illustration of the value of the combined compile-time and run-time approach we have been discussing. It has only 2 distinct profiles from the Lapack 77 test cases, and analysis of the basic-block coverage reveals that neither covers cases where N has a value greater than one. However, the Lapack 95 tests do exercise those cases and reveal a work-array dimension error because of the incorrect calculation of LWMIN.

The original 1.3 million data sets generated were reduced to only 6.5K, producing an equivalent basic-block coverage of the top level routines. The data extraction process was not perfect in that we only extracted data which generated unique execution profiles for the user callable routines. We would thus expect a reduction in the overall coverage due to the fact that two identical profiles to a high level routine generated distinct profiles at a lower level.

Having instrumented the Lapack routines using LDRA Testbed the driver programs were run using all the extracted datasets. Our experience of using this package was that the initial effort of instrumenting and running the bulk of the reduced data to analyse the execution histories took several overnight runs on a 1GHz Pentium 3 with 256K RAM. Adding new datasets is then relatively cheap taking just a few minutes per dataset. We obtained the following coverage metric values with the reduced data sets

1. basic-block coverage: 80%
2. branch coverage: 72%
3. LCSAJ: 32%

These results show that we have lost about 7% coverage of the basic blocks in the secondary level routines. At this level it is worthwhile upgrading our extraction process in order to associate the secondary level profiles with the top level data sets that generated them. This will allow the checking of results to take place at a higher level. It is still most likely that we will not be able to achieve 100% basic-block coverage without making independent calls to second level routines, although, in this case, there is no way of knowing whether such calls are possible using legal data at the user callable routine level. Additionally it is possible that some paths and blocks of code can never be executed; in some instances it may be possible to identify dead code and remove it. However there may be circumstances where unexecuted code is the result of defensive programming and in these cases the statements should certainly not be removed. It would be useful (and possibly interesting) if the user could be prompted to submit data causing execution of these sections of code for inclusion in the test data.

The achieved branch coverage is lagging a further 8% behind the basic-block coverage and the LCSAJ metric is well below the recommended level of 60%. Further work is needed with individual routines where LCSAJ coverage is particularly low from the reduced datasets in order to determine whether it is significantly better with the full complement of randomly generated datasets.

While most of the work we have reported here has focussed on the Lapack 77 testing strategy, we are also actively looking at the Lapack 95 test routines in order to compare the coverage they provide. It is interesting to note that the addition of compile-time and run-time checks to the test routines and example programs supplied with that version of the package reveal many problems simlar to those thrown up with the 77 version: unchecked use of optional arguments (SGELSD_F90); deallocation of unallocated space (SSPGVD_F90); use of an incorrect-precision routine (SCHKGE); incorrect array sizes (LA_TEST_SGBSV, SERRGE and many others); use of literals as INOUT parameters (SERRVX); use of unset OUT parameters (LA_TEST_SGECON); incorrect intent (LA_TEST_SGELSS); incorrect slicing (LA_TEST_SGESDD); floating-point overflow (LA_TEST_SPPSV).

5 Conclusions

Particular testing strategies should be chosen to fit specific goals and it is important to appreciate that the existing test strategy of the Lapack suite is strongly influenced by the desire to check portability rather than code coverage. A feature of the existing testing strategy is to batch up tests in two ways

– using multiple data sets on a single routine

– using similar data on multiple routines.

It was discovered that the very act of batching up tests allowed some errors to be masked — typically through uninitialised variables used in later tests having been set by earlier tests. Thus, while it requires far more organization, there are very definite advantages to be gained from running each test separately. An approach using separate tests also supports the incremental approach to improving testing metrics along with the introduction of additional tests whenever a routine is enhanced, or a new error is discovered and then corrected. Such additional tests serve as regression tests for future releases.

The need for dealing with a large number of test cases has led us to develop a more flexible testing environment which allows for the easy documentation and addition of test cases while keeping track of the code coverage obtained. Ideally, we would aim to extend the existing test suite in order to achieve 100% basic-block coverage as well as increasing the metric values obtained for branch and conditional coverage. Our major problem at present is finding data that will exercise the remaining unexecuted statements. The calling structure of the numerical routines is hierarchical and, in many cases it is clear that unexecuted code in higher level routines would only be executed after the failure of some lower level routine. It is not clear whether such situations are actually known to occur or if the higher level code is the result of defensive programming. In the former case it might be useful to issue a message encouraging the user to submit such data to the project team. While defensive programming practices are obviously not to be discouraged, they can potentially confuse the picture when considering how thorough a test suite is in practice and defensive code should be well documented within the source code.

We have shown how the use of software testing metrics may be used to provide a quantitative measure of how good the testing process actually is as a confidence boosting measure of program correctness.

We have set up a framework for testing the Lapack suite of routines in terms of a highly targetted reduced collection of datasets. Even so we are still some way from achieving 100% basic-block coverage which is considered to be the weakest coverage metric. Indeed Beizer [4] has argued that even if complete branch coverage is achieved then probably less than 50% of the faults left in the released software will have been found. We will be endeavouring to increase the basic-block coverage as well as improving the condition and branch coverage and LCSAJ metrics.

References

1. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK: users' guide*. SIAM, Philadelphia, third edition, 1999.
2. V. A. Barker, L. S. Blackford, J. Dongarra, J. Du Croz, S. Hammarling, M. Marinova, J. Waśniewski, and P. Yalamov. *LAPACK95: Users' Guide*. SIAM, Philadelphia, 2001.
3. D.J. Barnes and T.R. Hopkins. The evolution and testing of a medium sized numerical package. In H.P. Langtangen, A.M. Bruaset, and E. Quak, editors, *Advances in Software Tools for Scientific Computing*, volume 10 of *Lecture Notes in Computational Science and Engineering*, pages 225–238. Springer-Verlag, Berlin, 2000.

4. B. Beizer. *Software System Testing and Quality Assurance*. Van Nostrand Reinhold, New York, US, 1984.
5. John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal Linear Algebra Methods Environment. *ACM Transactions on Mathematical Software*, 27(4):422–455, December 2001.
6. Kelly J. Hayhurst, Dan S. Veerhusen, John J. Chilenski, and Leanna K. Rierson. A practical tutorial on modified condition/decision coverage. Technical Report TM-2001-210876, NASA, Langley Research Center, Hampton, Virginia 23681-2199, May 2001.
7. Tim Hopkins. A comment on the presentation and testing of CALGO codes and a remark on Algorithm 639: To integrate some infinite oscillating tails. *ACM Transactions on Mathematical Software*, 28(3):285–300, September 2002.
8. ISO, Geneva, Switzerland. *ISO/IEC 9899:1990 Information technology - Programming Language C*, 1990.
9. ISO/IEC. *Information Technology – Programming Languages – Fortran - Part 1: Base Language (ISO/IEC 1539-1:1997)*. ISO/IEC Copyright Office, Geneva, 1997.
10. C. Kaner, J. Falk, and H.Q. Nguyen. *Testing Computer Software*. Wiley, Chichester, UK, 1999.
11. Lahey Computer Systems, Inc., Incline Village, NV, USA. *Lahey/Fujitsu Fortran 95 User's Guide*, Revision C edition, 2000.
12. LDRA Ltd, Liverpool, UK. *LDRA Testbed: Technical Description v7.0*, 2000.
13. Numerical Algorithms Group Ltd., Oxford, UK. *NAGWare Fortran Tools (Release 4.0)*, September 1999.
14. Numerical Algorithms Group Ltd., Oxford, UK. *NAGWare f95 Compiler (Release 5.0)*, November 2003.
15. A.J. Offut, A. Lee, G. Rothermel, R.H. Untch, and C. Zapf. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering Methodology*, 5(2):99–118, April 1996.
16. Sun Microsystems , Inc., Santa Clara, CA. *Fortran User's Guide (Forte Developer 7)*, Revision A edition, May 2002.

A Matrix-Type for Performance–Portability

N. Peter Drakenberg

Department of Microelectronics and Information Technology
The Royal Institute of Technology
Stockholm, Sweden

Abstract. When matrix computations are expressed in conventional programming languages, matrices are almost exclusively represented by arrays, but arrays are also used to represent many other kinds of entities, such as grids, lists, hash tables, etc. The responsibility for achieving efficient matrix computations is usually seen as resting on compilers, which in turn apply loop restructuring and reordering transformations to adapt programs and program fragments to target different architectures. Unfortunately, compilers are often unable to restructure conventional algorithms for matrix computations into their block or block-recursive counterparts, which are required to obtain acceptable levels of perfomance on most current (and future) hardware systems.

We present a datatype which is dedicated to the representation of dense matrices. In contrast to arrays, for which index-based element-reference is the basic (primitive) operation, the primitive operations of our specialized matrix-type are composition and decomposition of/into submatrices. Decomposition of a matrix into submatrices (of unspecified sizes) is a key operation in the development of block algorithms for matrix computations, and through direct and explicit expression of (ambiguous) decompositions of matrices into submatrices, block algorithms can be expressed explicitly and at the same time the task of finding good decomposition parameters (i.e., block sizes) for each specific target system, is exposed to and made suitable for compilers.

1 Introduction

In current practice, computational software needs to be more or less extensively adapted to and tuned for each kind of target platform it will run on in order to satisfy performance expectations. In contrast, performance–portable software would achieve high performance on different kinds of target platforms *without* human involvement in target-specific tuning and optimization, and is therefore clearly desirable.

In mainstream software development, adapting and tuning programs to target platforms is seen as the responsibility of compilers, and significant research efforts have been devoted to the automatic application of reordering and restructuring program transformations by compilers (e.g., see [1,2] and references therein). Unfortunately, such methods are unable to derive important rephrasings of key algorithms in matrix computations [3,4] and have for the most part also not been successful in automatic parallelization for hardware systems with distributed memory. One of the main reasons behind these difficulties is that in typical programming languages it is not possible to express how to perform a computation, with sufficient ambiguity to allow compilers to adapt such computations in different ways to systems with different hardware characteristics.

As a concrete example to illustrate our notion of ambiguity as it pertains to dense matrices, we consider the operation of multiplying matrices. As is well known, efficient matrix computations rely on so-called block-algorithms, which perform their computations on appropriately sized sub-matrices rather than complete matrices. For conventional matrix multiplication:

```
do i = 1,M
   do j = 1,N
      do k = 1,P
         C(i,j) = C(i,j) + A(i,k) * B(k,j)          (1.1)
      enddo
   enddo
enddo
```

the transformation into a corresponding block algorithm:

```
do it = 1,M step T1
   do jt = 1,N step T2
      do kt = 1,P step T3
         do i = it,it+l-1
            do j = jt,jt+l-1
               do k = kt,kt+l-1
                  C(i,j) = C(i,j) + A(i,k) * B(k,j)     (1.2)
               end do
            end do
         end do
      end do
   end do
end do
```

is known as loop-tiling or unroll-and-jam [1,2] and is applied to simply structured program fragments, such as matrix multiplication, by many restructuring compilers (e.g., see [5,6]). However, the block algorithms of choice for QR factorization [7] and LU factorization with partial pivoting cannot be derived through program restructuring transformations [3,4]. In practice, therefore, authors of programs with portability requirements that exclude the option of maintaining different program sources for different target architectures are faced with the alternatives of either writing code of the kind shown in (1.1) and rely on compiler transformations to meet performance requirements, or writing code of the kind shown in (1.2) and decide upon fixed values for T_1, T_2, T_3 which will lead to sub-optimal performance on many systems and/or very poor performance for some matrix sizes [8]. The alternative of calculating suitable values for T_1, T_2, T_3 at run-time has also been investigated [9], but is rarely if ever adopted in practice because of the difficulty of finding appropriate and fast functions that compute suitable such values.

2 A Dedicated Dense–Matrix Type

Using arrays to implement matrices, as is usually the only reasonable alternative in most programming languages, has the drawbacks that it is difficult or impossible for compilers to reliably and precisely infer storage demands and usage across non-trivial program fragments (as witnessed by the work-region arguments of LAPACK [10] functions/subroutines), and that as mentioned above, there are no natural means of expressing block matrix algorithms abstractly (i.e., without explicitly specifying block sizes). In addition, the use of arrays has its usual performance issues of possibly needing costly bounds checking on indices to ensure correctness and that the program analyses needed for aggressive loop and array usage optimizations require very simply structured program fragments and simple array index expressions [11,12,13]. The option of explicitly calculating block-sizes at run-time, mentioned above, will frequently inhibit both elimination of array bounds checking and array dependence analysis as it requires forms of analyses which are usually not implemented in compilers (e.g., [12]).

In contrast, the matrix-type which is the subject of this paper and which we now describe, does not as its primary function provide a mapping from indices to values, nor is direct access to matrix elements through index values a "built-in" operation of these matrices. Instead, matrix elements are accessed by decomposition of matrices, similarly to how elements of lists are accessed in functional languages such as Haskell [14] and Standard ML [15]:

```
function ddot: (DM[1,n], DM[n,1]) -> Double Real is
   ddot(<a>, <b>) = a * b
   ddot( (1,2)::<a>|A:[1,*], (2,1)::<b>|B:[*,1] ) =
      let d = ddot(A, B) in a*b+d
end function
```

Despite the conceptual similarities with list-types illustrated by the example, matrices of the type being described are very different entities from lists. First of all and unlike lists, matrices are always two-dimensional (row- and column-vectors are seen as special cases of $m \times n$ matrices for which $m = 1$ and $n = 1$, respectively), and may be composed and/or decomposed with equal efficiency from all four (4) sides. Secondly, in order to ensure that otherwise potentially costly matrix composition- and decomposition-operations really can be performed efficiently, "behavioral" constraints are imposed on all program fragments that use or define (i.e., create) matrices. These constraints read as follows:

– More than one reference to the same matrix or to shared/common sub-matrices are never allowed to simultaneously leave a scope of definitions, such as `let`-constructs or function definitions.
– For each function that takes matrix argument(s) and yields matrix result(s), the dimensions (i.e., size) of each result matrix must be an affine (i.e., linear+constant) function of the dimensions of argument matrices that is inferable at compile-time.
– For all matrices involved in a composition, which do not have constant dimensions, it must be possible to preallocate memory for all run-time instances so as to let all compositions be performed without any copying of matrix elements.

For ordinary arrays, constraints corresponding to those above would impose sufficiently strong limitations as to render such arrays essentially useless for everything except possibly dense matrix computations. Furthermore, finding ways to reliably and reasonably impose such constraints on uses of ordinary arrays would be fairly difficult.

In addition to the patterns for decomposition of matrices shown above and their obvious extension to general $m \times n$ matrices (e.g., using a pattern such as

```
(2,2)::<a>|b:[1,⋆]|c:[⋆,1]|D:[⋆,⋆],
```

to decompose an $m \times n$ matrix into a scalar `a`, row-vector `b`, column-vector `c` and an $(m-1) \times (n-1)$ matrix `D`), it is rather natural to permit ambiguous decompositions such as

```
(2,2)::A:[⋆,⋆]|B:[⋆,⋆]|C:[⋆,⋆]|D:[⋆,⋆],
```

according to which an $m \times n$ matrix is decomposed into the submatrices `A`, `B`, `C`, and `D`, such that $m_{\mathtt{A}} = m_{\mathtt{B}}, m_{\mathtt{C}} = m_{\mathtt{D}}, n_{\mathtt{A}} = n_{\mathtt{C}}, n_{\mathtt{B}} = n_{\mathtt{D}}$ and $m_{\mathtt{A}} + m_{\mathtt{C}} = m, n_{\mathtt{A}} + n_{\mathtt{B}} = n$. The constraints on matrix usage described above ensure that the dimensions of result matrices of functions remain the same for all specific choices of decompositions that are consistent with the ambiguous decomposition patterns used, and that storage for such result matrices can be preallocated ahead of time. Clearly, ambiguous matrix decompositions make it possible to explicitly express block matrix algorithms while leaving block-sizes open for compilers to select so as to best suit each specific hardware target.

Before looking more closely at how matrix algorithms would be expressed using the just described matrix-type and decomposition constructs, we mention that the meanings of the latter are not really intuitively and immediately obvious when written as plain text. Current integrated development environments (IDEs), such as for example Eclipse and NetBeans, could clearly be extended to support entry and display of matrix expressions in closer agreement with established typographical conventions. For this reason, we henceforth use bold capital letters as names of $m \times n$ matrices for arbitrary $m, n \geq 1$, bold non-capital letters as names of row- and column-vectors, and non-bold letters as names of scalar values. The decomposition patterns

```
(2,2)::<a>:[1,1]|b:[1,⋆]|c:[⋆,1]|D:[⋆,⋆]
```

and

```
(2,2)::A:[⋆,⋆]|B:[⋆,⋆]|C:[⋆,⋆]|D:[⋆,⋆]
```

are presented/displayed as

$$\begin{bmatrix} \langle a \rangle & \mathbf{b} \\ \mathbf{c} & \mathbf{D} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix},$$

respectively, for which the intended meanings when used both as decomposition patterns composition expressions, are intuitively obvious.

3 Expressing Matrix Computations

Our first example of a matrix algorithm expressed using the matrix-type described in the previous section is used (by other functions) to compute the Cholesky factorization of small matrices and is defined as follows:

```
function $\mathtt{dpotf2}^{U}$: (DM[*,*], DM[*,*], DM[*,*]) -> DM[*,*] is
   $\mathtt{dpotf2}^{U}$( [<e>], $\mathbf{x}$, $\mathbf{R}$ ) =
      let t = e - $\mathtt{ddot}^{TN}$($\mathbf{x}$, $\mathbf{x}$) ;
          $\mathbf{z}^{t}$ = zeros(transpose($\mathbf{x}$))
      in $\begin{bmatrix} \mathbf{R} & \mathbf{x} \\ \mathbf{z}^{t} & <\sqrt{t}> \end{bmatrix}$

   $\mathtt{dpotf2}^{U}$( $\begin{bmatrix} <e> & \mathbf{v}^{t} \\ \mathbf{w} & \mathbf{M} \end{bmatrix}$, $\begin{bmatrix} \mathbf{x} & \mathbf{A} \end{bmatrix}$, $\mathbf{R}\hat{E}$) =
      let t = $\sqrt{e - \mathtt{ddot}^{TN}(\mathbf{x}, \mathbf{x})}$ ;
         $\mathbf{y}$ = $\mathtt{dgemv}^{T}$( -1.0, $\mathbf{A}$, $\mathbf{x}$, 1.0, $\mathbf{v}^{t}$ ) ;
         $\mathbf{u}^{t}$ = dscal( 1.0/t, transpose($\mathbf{y}$) ) ;
         $\mathbf{z}^{t}$ = zeros( transpose($\mathbf{x}$) )
      in $\mathtt{dpotf2}^{U}$( $\mathbf{M}$, $\begin{bmatrix} \mathbf{A} \\ \mathbf{u}^{t} \end{bmatrix}$, $\begin{bmatrix} \mathbf{R} & \mathbf{y} \\ \mathbf{z}^{t} & <t> \end{bmatrix}$ )
end function
```

The third argument of dpotf2 is used as an accumulator of partial results, and is the main contributor to the result returned when the first two arguments of dpotf2 is a scalar and a column vector, respectively. In the definition of the dpotf2 function we have used superscripted letters in names to add clarity (i.e., $\mathbf{z}^{t}$, $\mathbf{v}^{t}$ and $\mathbf{u}^{t}$ are all row-vectors) and so as not to prevent the matrix usage constraints from being enforced. Specifically, character arguments to BLAS functions/subroutines which typically have far-reaching implications on the actual computations performed (e.g., by indicating whether matrix arguments should be transposed or not), and therefore such character arguments have been incorporated into the names of functions by using superscript letters (e.g., $\mathtt{dgemv}^{T}$ and $\mathtt{dgemv}^{N}$, within which the argument matrix is transposed and not transposed, respectively, when forming matrix–vector products). With the exception of zeros, transpose and the use of superscript letters in the manner just described, all functions mentioned in the definition of dpotf2 correspond to identically named BLAS functions/subroutines [16,17,18].

The dpotf2 function would not be suitable to export from a linear algebra package, as it may easily be mistakenly used in erroneous ways. Figure 1 instead shows several different definitions of functions for Cholesky factorization that are more suitable to export, and of which one (c) is a block-algorithm that is expressed using ambiguous decompositions. The same conventions as used in the definition of dpotf2 regarding superscript letters and names of BLAS functions have been used in Figure 1, and in addition four different variants of the ddot-function are assumed to exist for which argument vectors are either transposed or untransposed, respectively, as indicated by the superscripted letters that follow ddot in their names.

4 Experimental Evaluation

The current version of our compiler translates a syntactically less pleasant but otherwise identical language to that described above, by the name of core-Aslan, into the IF1 representation [20] of the Optimizing Sisal compiler (OSC). The advantages gained by

```
function cholesky: (DM[*,*]) -> (DM[*,*]) is
  cholesky([<a>]) = [<$\sqrt{a}$>]
  cholesky($\begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c}^t & <s> \end{bmatrix}$) =
    let $\mathbf{R}$ = cholesky($\mathbf{A}$)
        $\mathbf{v}$ = trsv$^{UNN}$($\mathbf{R}$, $\mathbf{b}$)
        t = s - ddot$^{TN}$($\mathbf{v}$, $\mathbf{v}$)
        $\mathbf{z}^t$ = zeros($\mathbf{c}^t$)
    in $\begin{bmatrix} \mathbf{R} & \mathbf{v} \\ \mathbf{z}^t & <\sqrt{t}> \end{bmatrix}$
end cholesky
```

(a) Plain (textbook) Cholesky factorization

```
function cholesky: (DM[*,*]) -> (DM[*,*]) is
  cholesky([<a>]) = [<$\sqrt{a}$>]
  cholesky($\begin{bmatrix} <s> & \mathbf{b}^t \\ \mathbf{c} & \mathbf{A} \end{bmatrix}$) =
    let t = $\sqrt{\text{s - ddot}^{NT}(\mathbf{b}^t, \mathbf{b}^t)}$
        $\mathbf{u}^t$ = dscal(1.0/t, $\mathbf{b}^t$)
    in dpotf2$^{U}$($\mathbf{A}$, $\mathbf{u}^t$, [<t>])
end cholesky
```

(b) Pointwise Cholesky from LAPACK [10]

```
function cholesky: (DM[*,*]) -> (DM[*,*]) is
  cholesky([<a>]) = [<$\sqrt{a}$>]
  cholesky($\begin{bmatrix} <s> & \mathbf{b}^t \\ \mathbf{c} & \mathbf{A} \end{bmatrix}$) =
    let t = $\sqrt{\text{s - ddot}^{NT}(\mathbf{b}^t, \mathbf{b}^t)}$
        $\mathbf{u}^t$ = dscal(1.0/t, $\mathbf{b}^t$)
    in dpotf2$^{U}$([<t>], $\mathbf{u}^t$, $\mathbf{A}$)
  cholesky($\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$) =
    let $\mathbf{U}$ = cholesky($\mathbf{A}$)
        $\mathbf{V}$ = trsm$^{LUTN}$(1.0, $\mathbf{U}$, $\mathbf{B}$)
        $\mathbf{W}$ = syrk$^{UT}$(-1.0, $\mathbf{V}$, 1.0, $\mathbf{D}$)
        $\mathbf{X}$ = cholesky( $\mathbf{W}$ )
        $\mathbf{Z}$ = zeros( $\mathbf{C}$ )
    in $\begin{bmatrix} \mathbf{U} & \mathbf{V} \\ \mathbf{Z} & \mathbf{X} \end{bmatrix}$
end cholesky
```

(c) Block-recursive Cholesky factorization [19]

Fig. 1. Pretty-printed forms of three algorithms for Cholesky factorization

generating IF1-format representations of programs are that the analyses and optimizations performed by OSC, such as update-in-place analysis, copy elimination, and array pre-allocation [21] are applied to our programs without us having had to implement these analyses and optimizations.

We have expressed the subroutines `dgemm`, `dsyrk`, `dgemv` and `dtrsm` from level-2 (only `dgemv`) and level-3 BLAS using the matrix-type, composition and decomposition constructs, and other language elements described above. The performance achieved by these core-Aslan implementations was subsequently measured and compared against that achieved by the corresponding subroutines from the hardware vendor's (SGI) hand-tuned BLAS library.

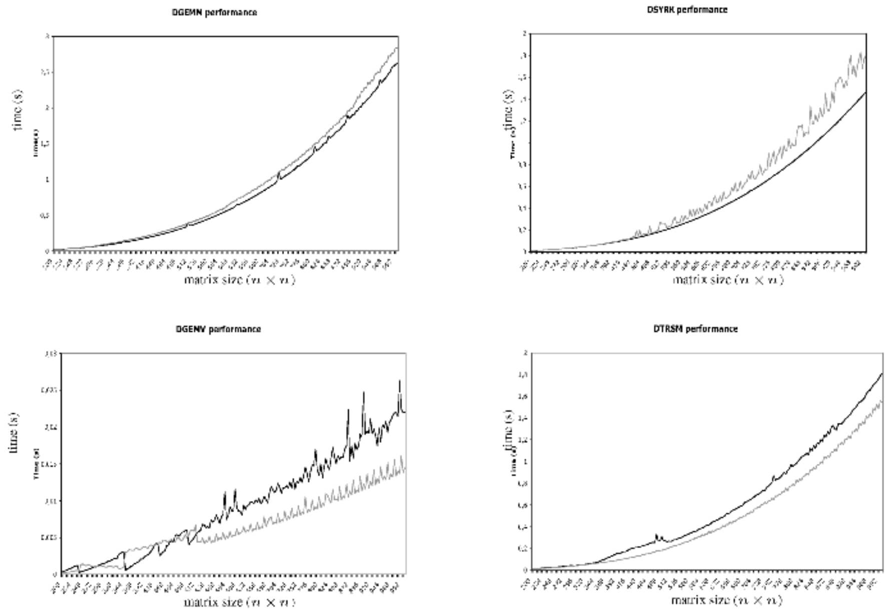


Fig. 2. Execution times of core-Aslan (black lines) and vendor hand-tuned (gray lines) implementations of the BLAS operations `dgemm`, `dsyrk`, `dgemv` and `dtrsm`

The diagrams in Figure 2 show execution times vs. matrix dimensions, measured on an SGI Octane$_2$ system[1], for our core-Aslan implementations of `dgemm`, `dsyrk`, `dgemv` and `dtrsm` (black lines) and corresponding measurements for the `dgemm`, `dsyrk`, `dgemv` and `dtrsm` functions from the hardware vendor's hand-tuned implementation

[1] The machine that was used for our measurements was equipped with 2 R12000 processors with operating frequencies of 400 MHz, each of which has a 32 kbyte 2-way set-associative L1 data cache, a 32 kbyte 2-way set-associative L1 instruction cache, and a 2 megabyte 2-way set-associative unified L2 cache. The two processors of the system share a 1 Gbyte main memory. No attempts towards parallel or multithreaded execution were made, and at all times the 2nd processor was unused with the exception of possible kernel/daemon-process activity.

of the BLAS-library (gray lines). As can be seen in Figure 2, the executable code generated from our more abstractly expressed BLAS functions is for the most part highly competitive with that of the presumably thoroughly optimized and tuned versions provided by the hardware vendor. The most notable case of worse performance (longer excecution times) for the core-Aslan implementation than for the hardware vendor's corresponding BLAS subroutine is observed for `dgemv`. However, as mentioned by Agarwal *et al.* in [22] and others, high-performance matrix–vector multiplication relies heavily on data-prefetching, but current compiler back-ends (which also the core-Aslan compiler depend on at present), are for the most part not able to insert prefetch operations with sufficient accuracy not to disturb the reuse of data already present in cache, and for this reason compiler/code-generator directives and/or flags to enable prefetching have not been used.

5 Related Work

To our knowledge, only programming languages embedded in numerical and symbolic computation environments, such as MATLAB$^{\mathrm{TM}}$, Maple$^{\mathrm{TM}}$, etc. have provided specific matrix-types. However, these matrix-types have been designed primarily to be reasonably convenient for interactive use rather than to promote and ensure efficiency in a compiled context. In contrast to the situation for matrix computations, the languages FIDIL [23], ZPL [24] and Titanium [25] all have types suitable for computational solution of partial differential equations.

In addition to the systems and languages mentioned above, languages and systems have been developed to support systematic design and implementation of high-performance matrix algorithms for cross-platform deployment. However, these languages and/or systems are either essentially languages for specification rather than implementation of algorithms [26], or systems that search the space of target-system related tuning parameters (e.g., software pipelining, prefetching, register blocking, cache blocking, etc., parameters) using a generate-and-measure methodology (a.k.a. empirical optimization) to find optimal combinations of parameter values [27]. None of these systems depart significantly from the model of matrices provided by widely used programming languages and the issue of how to reasonably expose tuning parameters to compilers is not addressed.

6 Conclusions

We have demonstrated that dense matrix computations can be expressed at higher levels of abstraction than usual and still achieve highly competitive performance. We show that through careful design of a specific data-type for dense matrices it is possible to express dense matrix computations so that target dependent tuning parameters can be identified and understood by compilers, and compilers can thereby potentially maintain performance–portability of such software across a wide range of different target systems, without any need for manual target-specific tuning.

Having a distinct type for dense matrices in languages intended for scientific computations has the advantages that it is feasible to impose stricter usage constraints and these in turn enable more precise and extensive analyses and optimization of run-time behavior. Furthermore, it becomes feasible also in practice to use different storage layouts for different kinds of aggregate data, and much more natural to provide distinct and specialized types for sparse matrices and other kinds of irregular data whose properties and run-time behaviors are otherwise very difficult (i.e., effectively impossible) for compilers to determine.

References

1. Allen, R., Kennedy, K.: Optimizing Compilers for Modern Architectures: A Dependence-Based Approach. Morgan Kaufmann Publishers, San Francisco, California (2002)
2. Wolfe, M.: High Performance Compilers for Parallel Computing. Addison-Wesley, Redwood City, California (1996)
3. Carr, S., Kennedy, K.: Compiler blockability of numerical algorithms. In: Proc. Supercomputing'92, Minneapolis, Minn. (1992) 114–124
4. Carr, S., Lehoucq, R.B.: Compiler blockability of dense matrix factorizations. ACM Trans. Math. Softw. **23** (1997) 336–361
5. Sarkar, V.: Automatic selection of high-order transformations in the IBM XL FORTRAN compilers. IBM J. Res. Develop. **41** (1997) 233–264
6. Wolf, M., Maydan, D., Chen, D.K.: Combining loop transformations considering caches and scheduling. In: Proc. 29th IEEE/ACM Intl. Symp. on Microarchitecture, Paris, France (1996)
7. Schreiber, R., Van Loan, C.: A storage efficient WY representation for products of householder transformations. SIAM J. Scientific and Statistical Computing **10** (1989) 53–57
8. Lam, M.S., Rothberg, E.E., Wolf, M.E.: The cache performance and optimizations of blocked algorithms. In: Proc. 4th Int. Conf. on Architectural Support for Programming Languages and Operating Systems. (1991) 63–74
9. Coleman, S., McKinley, K.S.: Tile size selection using cache organization and data layout. In: Proc. Conf. on Prog. Lang. Design and Implementation, La Jolla, CA (1995) 279–290
10. Anderson, E., et al.: LAPACK Users' Guide. 2nd edn. SIAM, Philadelphia, PA 19104-2688 (1995)
11. Feautrier, P.: Dataflow analysis of scalar and array references. Int. J. Parallel Prog. **20** (1991) 23–53
12. Maslov, V., Pugh, W.: Simplifying polynomial constraints over integers to make dependence analysis more precise. Technical Report UMIACS-CS-TR-93-68.1, Dept. of Computer Science, University of Maryland, College Park, MD 20742 (1994)
13. Barthou, D., Collard, J.F., Feautrier, P.: Fuzzy array dataflow analysis. J. Parall. Distr. Comput. **40** (1997) 210–226
14. Peyton Jones, S., Hughes, J. (Eds.): Report on the programming language Haskell 98. http://www.haskell.org/report (1998)
15. Milner, R., Tofte, M., Harper, R., MacQueen, D.: The Definition of Standard ML. Revised edn. The MIT Press, Cambridge, Massachusetts (1997)
16. Lawson, C., Hanson, R., Kincaid, R., Krogh, F.: Basic linear algebra subprograms for Fortran usage. ACM Trans. Math. Softw. **5** (1979) 308–323
17. Dongarra, J., Du Croz, J., Hammarling, S., Hansson, R.: An extended set of Fortran basic linear algebra subprograms. ACM Trans. Math. Softw. **14** (1988) 1–17, 18–32
18. Dongarra, J., Du Croz, J., Duff, I., Hammarling, S.: A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Softw. **16** (1990) 1–17
19. Gustavson, F.G.: Recursion leads to automatic blocking for dense linear-algebra algorithms. IBM J. Res. Develop. **41** (1997) 737–755

20. Skedzielewski, S., Glauert, J.: IF1 An intermediate form for applicative languages. Technical Report M-170, Lawrence Livermore National Laboratory, Livermore, CA 94550 (1985)
21. Cann, D.C., Evripidou, P.: Advanced array optimizations for high performance functional languages. IEEE Trans. Parall. Distr. Sys. **6** (1995) 229–239
22. Agarwal, R.C., Gustavson, F.G., Zubair, M.: Improving performance of linear algebra algorithms for dense matrices, using algorithmic prefetch. IBM J. Res. Develop. **38** (1994) 265–275
23. Hilfinger, P.N., Colella, P.: FIDIL: A language for scientific programming. In: R. Grossman, editor, Symbolic Computing: Applications to Scientific Computing, Society for Industrial and Applied Mathematics, Philadelphia (1989), 97–138
24. Lin, C.: ZPL Language reference manual. Technical Report 94-10-06, Dept. of Computer Science, University of Washington (1994)
25. Yelick, K. *et al.*, Aiken A.: Titanium: A high-performance Java dialect. Concurrency: Practice and Experience **10** (1998) 825–836
26. Gunnels, J.A., Gustavson, F.G., Henry, G.M., van de Geijn, R.A.: Flame: Formal linear algebra methods environment. ACM Trans. Math. Softw. **27** (2001) 422–455
27. Whaley, R.C., Petitet, A., Dongarra, J.J.: Automated empirical optimization of software and the ATLAS project. Parallel Computing **27** (2001) 3–35
28. Demmel, J.W., Dongarra, J.J., *et al.*: Self-adapting linear algebra algorithms and software. Proc. IEEE, Special issue on program generation, optimization, and adaptation (2005) to appear. See also http://bebop.cs.berkeley.edu/pubs/ieee_sans.pdf

A New Array Format for Symmetric and Triangular Matrices

John A. Gunnels and Fred G. Gustavson

IBM T.J. Watson Research Center, Yorktown Heights NY 10598, USA

Abstract. We introduce a new data format for storing triangular and symmetric matrices called HFP (Hybrid Full Packed). The standard two dimensional arrays of Fortran and C (also known as full format) that are used to store triangular and symmetric matrices waste half the storage space but provide high performance via the use of level 3 BLAS. Packed format arrays fully utilize storage (array space) but provide low performance as there are *no* level 3 packed BLAS. We combine the good features of packed and full storage using HFP format to obtain high performance using L3 (Level 3) BLAS as HFP is totally full format. Also, HFP format requires exactly the same minimal storage as packed storage. Each LAPACK full and/or packed symmetric/triangular routine becomes a single new HFP routine. LAPACK has some 125 times two such symmetric/triangular routines and hence some 125 new HFP routines can be produced. These new routines are trivial to produce as they merely consist of calls to existing LAPACK routines and Level 3 BLAS. Data format conversion routines between the conventional formats and HFP are briefly discussed as these routines make existing codes compatible with the new HFP format. We use the LAPACK routines for Cholesky factorization and inverse computation to illustrate this new work and to describe its performance. Performance of HPF verses LAPACK full routines is slightly better while using half the storage. Performance is roughly one to seven times faster for LAPACK packed routines while using the same storage. Performance runs were done on the IBM Power 4 using only existing LAPACK routines and ESSL level 3 BLAS.

1 Introduction

We introduce a new matrix storage format called HFP (Hybrid Full Packed). We will cover six points about current LAPACK [1999] and HFP [2004] storage formats as well as seeing how general the HFP format is for LAPACK routines that operate on symmetric / triangular matrices.

Full format triangular and symmetric arrays waste half the storage but provide high performance for LAPACK via the use of level 3 BLAS. By full format we mean the standard 2D arrays of Fortran and C that are used by LAPACK routines to store these matrices. Packed format arrays fully utilize storage but provide low performance as there are *no* level 3 packed BLAS. An LAPACK example of this is its routines DPOTRF / DPPTRF which compute the Cholesky factorization of a positive definite symmetric matrix. This paper will demonstrate that it is possible combine the strong feature of both full and packed format by replacing both of them with the new HFP format.

HFP format is totally full format and hence it is possible to obtain high performance using L3 (Level 3) BLAS. Additionally HFP format requires exactly $NT = N(N+1)/2$

storage locations to represent an order N symmetric and/or triangular matrix. Thus an LAPACK full and/or packed symmetric/triangular routine becomes a single new HFP routine. There are some 125 times two such LAPACK symmetric/triangular routines and hence some 125 new routines can be produced.

The HFP format is possible because two order N isosceles triangles can be rearranged to form a rectangle of size N by $N+1$ or because an order N isosceles triangle and an order $N+1$ isosceles triangle can be rearranged to form a square of order $N+1$. Here an order N isosceles triangle is made up of NT unit squares.

Let A be an order N symmetric / triangular matrix. In Section 2 we give an example where N is odd and N is even. Covering the odd and even cases of N for both UPLO = ‘U’ and ‘L’ is general. However, because of symmetry there can be different ways to define HFP format. Our definition is essentially independent of uplo as are the example Cholesky and Inverse codes that operate on HPF format. These codes are given in Section 3.

HFP format consist of two full arrays concatenated along their common row dimension. Let $n1 = N/2$ and $n2 = N - n1$. The first full array T holds two isosceles triangles T1 and T2 of sizes $n1$ and $n2$. For both uplo = ‘L’ and ‘U’ T1 is in lower format and T2 is in upper format. The second full array S1 contains the square or the near square between the two triangles T1 and T2. For uplo = ‘L’ S1 is stored in row major order (format is ‘Transpose’) and for uplo = ‘U’ S1 is stored in col major order (format is ‘Normal’). Together these two arrays occupy a single array space of size $2n1 + 1$ rows by $n2$ columns.

We now consider performance aspects of using HFP format in the context of using LAPACK routines on triangular matrices stored in HPF format. Let X be a level 3 LAPACK routine that operates either on standard packed or full format. X has a full L3 LAPACK block algorithm, call it FX. Write a simple related partition algorithm SRPA with partition sizes $n1$ and $n2$. Apply the new SRPA on the new HPF data structure. The new SRPA almost always has four major steps consisting entirely of calls to existing full format LAPACK routines in two steps and calls to level 3 BLAS in the remaining two steps:

```
call FX('L',n1,T1,ldt) ! step 1
call L3BLAS(n1,n2,'L',T1,ldt,S,lds) ! step 2
call L3BLAS(n1,n2,S,lds,'U',T2,ldt) ! step 3
call FX('U',n2,T2,ldt) ! step 4
```

Section 4 gives performance results for the four LAPACK routines detailed in Section 3, namely DPOTRF / DPPTRF(uplo=‘L’ and uplo = ‘U’) verses SRPA for DPOTRF / DPPTRF and for DPOTRI / DPPTRI(uplo=‘L’ and uplo = ‘U’) verses SRPA for DPOTRI / DPPTRI. Results were run on an IBM Power4 processor using ESSL Level 3 BLAS. In all cases, only LAPACK code was called so in a sense all that is being compared is standard LAPACK on different standard full and packed data formats verses SRPA. The results show that SRPA is slightly better than LAPACK full routines and roughly one to seven times faster than LAPACK packed routines.

The SRPA algorithm assumes the input matrix is in HFP format. We have produced eight routines for converting between packed / full format and HFP format, both inplace and out-of-place. The packed routines are called DPTHF and DHFTP. It is suggested that the eight routines be made available with LAPACK. This method described above is quite general. It offers the possibility to replace all LAPACK packed L2 codes and all LAPACK full L3 codes with new simple L3 codes based on existing LAPACK L3 full format codes. These new simple L3 codes are like the example code given above. The appeal is especially strong for SMP codes as many L3 BLAS are enabled for SMP implementations. Four routines offer efficient SMP implementations. Finally, if this new HFP format catches on as it might one can offer any user the ability to only use full format L2 BLAS. The user will not use the current packed L2 BLAS as HFP L2 BLAS are easily implemented by calls to full format L2 BLAS. These new codes will be efficient as no calls to DPTHF or DHFTP are needed as then the data format is already HFP.

2 Array Folding

We begin this section by reviewing standard packed format. It suffices to consider the order N of a packed array to be odd and even. And, it also suffices to consider specific values say $N = 9$ and $N = 8$ to describes what goes on generally for N odd and even. Recall first that a packed array stores it matrix elements column wise as concatenation of column vectors. For uplo = ‘U’ the vectors have lengths $1, 2, ..., N$ and for uplo =‘L’ the vectors have lengths $N, N - 1, ..., 1$. In the pictorial description of packed format below with $N = 9$ we show the i, j elements with zero origin and $0 <= i <= j < N$ for uplo=‘U’ and $N > i >= j >= 0$ for uplo=‘L’. To further elucidate, the 3,5 element of A which is depicted by 35 in the picture is in UP location 18 and the 5,3 element of A is in LP location 26. Both arrays UP and LP occupy locations 0:44; ie, 45 words of a linear array.

```
                   The case N = 9
               UP                                  LP
 00 01 02 03 04 05 06 07 08     00
    11 12 13 14 15 16 17 18     10 11
       22 23 24 25 26 27 28     20 21 22
          33 34 35 36 37 38     30 31 32 33
             44 45 46 47 48     40 41 42 43 44
                55 56 57 58     50 51 52 53 54 55
                   66 67 68     60 61 62 63 64 65 66
                      77 78     70 71 72 73 74 75 76 77
                         88     80 81 82 83 84 85 86 87 88
```

Both UHFP and LHFP formats consist of two full arrays. The first full array T holds two isosceles triangles T1 and T2 of sizes $n1$ and $n2$. The LDT of T is $n1 + 1$ and the number cols is $n2$. In the ‘L’ case T2 is stored in upper format so that the two isosceles triangles concatenate to form a compact square. In the ‘U’ case T1 is stored in lower format. Again T1 and T2 form a compact square. Now we remark about the storage location of the i, j element in a standard full 2D array A that is stored in column major order. Again, we are using 0 origin for both i and j. $A_{i,j}$ resides at location $i + lda * j$ in the array holding A. For example the 2,3 element in UHFP T and the 2,3 element in LHFT are both at location 2 + 5*3 = 17 of arrays UHFT and LHFT. According to the above mapping the elements are 6,7 of UP and 7,6 of LP and of course, due to symmetry, they are equal.

```
        UHFP T                              LHFP T
     44 45 46 47 48                      44 54 64 74 84
     00 55 56 57 58                      00 55 65 75 85
     01 11 66 67 68                      10 11 66 76 86
     02 12 22 77 78                      20 21 22 77 87
     03 13 23 33 88                      30 31 32 33 88
```

The second full array S1 contains the near square between the two triangles of sizes $n1$ and $n2$. For uplo = 'L' S1 is stored in row major order (format is 'Transpose') and for uplo = 'U' S1 is stored in col major order (format is 'Normal').

```
        UHFP S1                             LHFP S1
     04 05 06 07 08                      40 50 60 70 80
     14 15 16 17 18                      41 51 61 71 81
     24 25 26 27 28                      42 52 62 72 82
     34 35 36 37 38                      43 53 73 73 83
```

Together these two arrays (concatenation along the row dimension) occupy a single full array space of size $2n1 + 1$ rows by $n2$ columns. Note that LDS = $n1$ and that both arrays T and S1 are in column major format. The example is general for N odd and T + S1 hold exactly $NT = N(N + 1)/2$ elements as does the UP and LP arrays.

When N is even we get triangular arrays T1 and T2 of sizes $n1$ and $n2 = n1$. We give an example where $N = 8$. In the pictorial description of packed format below with $N = 8$ we show the i, j elements with zero origin and $0 <= i <= j < N$ for uplo='U' and $N > i >= j >= 0$ for uplo='L'. To further elucidate, the 3,5 element of A denoted by 35 in the picture is in UP location 18 and the 5,3 of element of A is in LP location 23. Both arrays UP and LP occupy locations 0:35; ie, 36 locations of a linear array.

```
                   The case N = 8
             UP                                   LP
 00 01 02 03 04 05 06 07        00
    11 12 13 14 15 16 17        10 11
       22 23 24 25 26 27        20 21 22
          33 34 35 36 37        30 31 32 33
             44 45 46 47        40 41 42 43 44
                55 56 57        50 51 52 53 54 55
                   66 67        60 61 62 63 64 65 66
                      77        70 71 72 73 74 75 76 77
```

Both UHFP and LHFP formats consist two full arrays. The first full array T holds two triangles T1 and T2 of sizes $n1$ and $n2$. The LDT is $n1 + 1$ and the number cols is $n2$. In the 'L' case T2 is stored in upper format so that the two triangles form a near compact square. In the 'U' case T1 is stored in lower format so that the two triangles form a near compact square.

```
        UHFP T                              LHFP T
     44 45 46 47                         44 54 64 74
     00 55 56 57                         00 55 65 75
```

```
01 11 66 67                              10 11 66 76
02 12 22 77                              20 21 22 77
03 13 23 33                              30 31 32 33
```

Now the 2,3 element in UHFP T and the 3,2 element in LHFT are both at location 2 + 5*3 = 17 of arrays UHFT and LHFT. According to the above mapping the elements are 6,7 of UP and 7,6 of LP and of course, due to symmetry they are equal. The second full array S1 contains the square between the two triangles. For uplo = 'L' S1 is stored in row major order (format is 'Transpose') and for uplo = 'U' S1 is stored in col major order (format is 'Normal').

```
    UHFP S1                                  LHFP S1
04 05 06 07                              40 50 60 70
14 15 16 17                              41 51 61 71
24 25 26 27                              42 52 62 72
34 35 36 37                              43 53 63 73
```

Together these two arrays (concatenation along the row dimension) form a single full array space of size $2n1 + 1$ rows by $n2$ columns. The example is general for N even and T + S1 hold exactly $NT = N(N + 1)/2$ elements as does the UP and LP arrays.

Let A be symmetric. Then, because of symmetry, the UHFP arrays for T and $S1$ are identical to the LHFP arrays for T and $S1$. Hence we identify UHFP with LHFP and this defines HFP format. Let A be triangular. Note that $L^T = U$ and $U^T = L$, so A has dual representations. In this case we define HFP as the UHFP or LHFP representation containing the nonzero part of A.

3 LAPACK Compatibility

In this section we consider two sets of LAPACK routines, DPOTRF/DPPTRF and DPOTRI/DPPTRI. Each set has two subcases uplo = 'L' and 'U'. Thus for each set consisting of four LAPACK codes we shall produce a single new code with exactly the same LAPACK function but which operates on the new HFP format described in section 2. Take any packed or full LAPACK routine. Both are based upon a full LAPACK L3 block algorithm that performs $O(N^3)$ operations. We write a Simple Related Partition Algorithm (SRPA) with partition sizes $n1$ and $n2$. Apply the new SRPA on the new HPF data structure. As our first example of this Section we take DPPTRF. The SRPA consists of 4 subroutine calls (steps 1 to 4) to existing L3 codes. Steps 1 and 4 are to DPOTRF to Cholesky factor matrices T1 and T2. Step 2 uses Cholesky factored T1 to compute the scaling of S1 by calling DTRSM. Then S1 is used to update T2 by calling DSYRK. Steps 1 to 4 are certainly not new. We are using a basic paradigm of the LAPACK library which is blocked based factorization. Here the blocking is two by two. Based on this remark it is should clear that this example is typical of the four step procedure given in the Introduction.

```
n1=n/2
n2=n-n1
it1=1 ! -> T1 in HPF
```

```
   it2=0 ! -> T2 in HPF
   ldt= n1+1 ! lda of both T1 and T2
   lds=n1 ! lda of S1
   is1=n2*(n1+1) ! -> S1 in HPF
 ! the routine dpthf converts packed format
 ! in AP to HFP format in AH
   call dpthf(ilu,N,AP,AH(it1),AH(it2),ldt, &
              AH(is1),lds)
   call dpotrf('L',n1,AH(it1),ldt,info) ! step 1
   if(info.gt.0)return
   call dtrsm('L','L','N','N',n1,n2,one,AH(it1), &
              ldt,AH(is1),lds) ! step 2
   call dsyrk('U','T',n2,n1,-one,AH(is1),lds, &
              one,AH(it2),ldt) ! step 3
   call dpotrf('U',n2,AH(it2),ldt,info) ! step 4
   if(info.gt.0)then
     info=info+n1
     return
   endif
 ! the routine dhftp converts HFP format in AH to
 ! packed format in AP
   call dhftp(ilu,N,AP,AH(it1),AH(it2),ldt,AH(is1),lds)
```

As our second example we take DPPTRI. Both subroutines DPOTRI/DPPTRI take the Cholesky factor, call it U, where $U^T \times U = A$ and computes $A^{-1} = V \times V^T$ where $V = U^{-1}$. This is done in two stages: Subroutine DTRTRI/DTPTRI computes V given triangular U and subroutine DLAUUM computes $V \times V^T$ given triangular V. Thus, following LAPACK there will be two related SRPAs. The first replaces DTRTRI/DTPTRI and each of these routines has two subcases for uplo='L' and 'U'. We now assume that above dhftp was NOT called so that T1, T2, and S1 are still in AH and pointers it1, it2 and is1 point to T1, T2, and S1. The SRPA code now follows:

```
  call dtrtri('L',DIAG,n1,AH(it1),ldt,info) ! step 1
  if(info.gt.0)return
  call dtrmm('L','L','T',DIAG,n1,n2,-one,AH(it1), &
             ldt,AH(is1),lds) !step 2
  call dtrtri('U',DIAG,n2,AH(it2),ldt,info) ! step 3
  if(info.gt.0)then
    info=info+n1
    RETURN
  endif
  call dtrmm('R','U','N',DIAG,n1,n2, one,AH(it2), &
             ldt,AH(is1),lds) !step 4
  RETURN
```

In the second stage we want the SRPA to replace the LAPACK routine DLAUUM:

```
  call DLAUUM('L',n1,AH(it1),ldt,info) ! step 1
  call dsyrk('L','N',n1,n2,one,AH(is1),lds, &
             one,AH(it1),ldt) !step 2
```

```
call dtrmm('R','U','T','N',n1,n2,one,AH(it2), &
           ldt,AH(is1),lds) !step 3
call DLAUUM('U',n2,AH(it2),ldt,info) ! step 4
RETURN
```

It should be quite clear now that the SRPAs are merely subroutine calls to existing LAPACK codes.

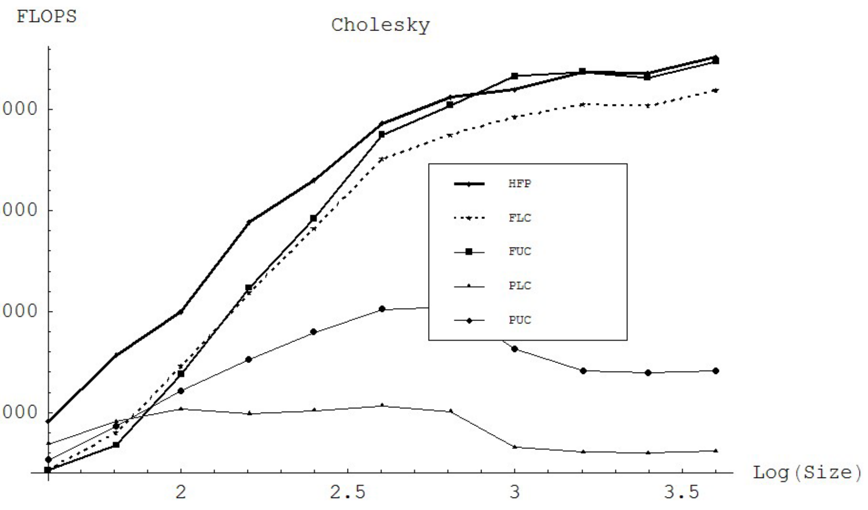
4 Performance Results

It is fairly obvious that the SRPAs of Section 3 operating on HFP format of Section 2 should perform about the same as the corresponding full format LAPACK routines. This is because both the SRPA code and the corresponding LAPACK code are the same and both data formats are full format. Also the SRPA code should outperform the corresponding LAPACK packed code by about the same margin as does the corresponding LAPACK full code. We give performance results for the IBM Power 4 Processor. The gain of full code over packed code is anywhere from roughly a factor of one to a factor of seven.

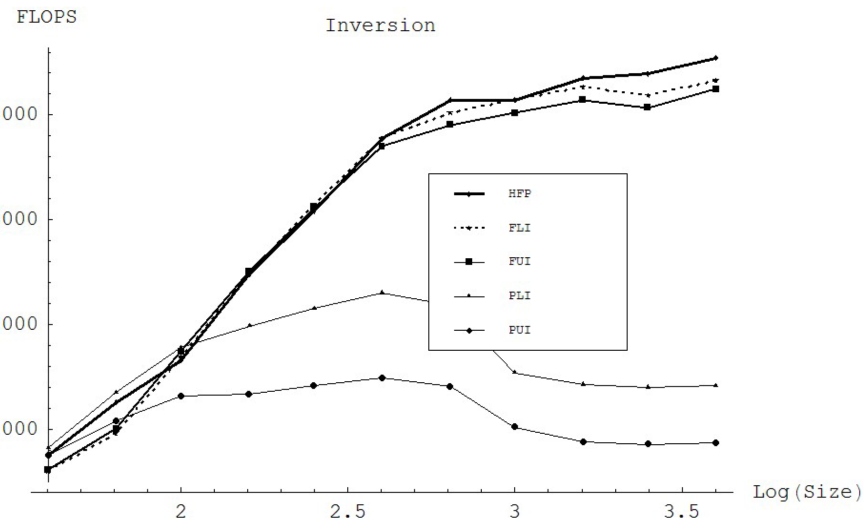
There are two performance graphs, one for Cholesky factor, suffix C and the other for Cholesky inverse, suffix I. For each graph we give four curves for LAPACK called FL, FU, PL, PU corresponding to Full,Uplo='L', Full,Uplo='U', Packed,Uplo='L', and Packed,Uplo='U' and a single curve HFP corresponding to SRPA. This is because the SRPA replaces each of these four LAPACK subroutines. Actually, we ran the SRPA routine four times and averaged their times. Performance is given in MFLOPS. We chose the order N of the matrices to follow a base 10 log distrbution. The values chosen were N = 40, 64, 100, 160, 250, 400, 640, 1000, 1600, 2500, and 4000. The corresponding log N values are 1.60, 1.81, 2, 2.20 2.40, 2.60, 2.81, 3, 3.20. 3.40, and 3.60.

As can be seen from Graph 1, SRPA performance is greater than FLC, PLC, and PUC performance. And SRPA performance is greater than FUC performance except at N = 1000 where FUC is 3 $\%$ faster. For graph two, with N $\geq$ 400 the SRPA curve is faster than the other four curves. For N $\leq$ 250 the performance ratios range from .92 to 2.18 (see Fig. 2). Returning to Graph 1, we see that SRPA is 1.33 to 7.35 times faster than PLC and 1.64 to 3.20 times faster than PUC. Similarly, for Graph 2, SRPA is .92 to 3.21 times faster than PLI and 1.01 to 5.24 times faster than PUI.

In Figure 2 we give performance ratios of HFP to the four LAPACK routines, FL, FU, PL, and PU. The row labeled HFP give MFLOPs values for that routine. To obtain the MFLOPs values for the other four routines, simply divide MFLOPs value by its associated ratio. For example, for $N = 640$, the MFLOPs for suffix C are 3818, 4042, 1011, 2041 and for suffix I, they are 4020, 3907, 2179, 1409.



(a)



(b)

Fig. 1. Absolute performance of algorithms (a) Cholesky Factorization. (b) Inversion

5 Conclusion

We have described a novel data format, HFP that can replace both standard full and packed formats for triangular and symmetric matrices. We showed that codes for the new data format HFP can be written by simply making calls to existing LAPACK routines and level 3 BLAS. Each new SRPA operating on HFP format data replaces two corresponding

N	40	64	100	160	250	400	640	1000	1600	2500	4000
HFP	914	1568	1994	2883	3302	3861	4123	4198	4371	4358	4520
FLC	2.12	1.97	1.37	1.32	1.17	1.10	1.10	1.07	1.08	1.08	1.08
FUC	2.14	2.32	1.45	1.29	1.13	1.03	1.02	.97	1.00	1.01	1.01
PLC	1.33	1.72	1.93	2.92	3.24	3.62	4.08	6.42	7.18	7.30	7.35
PUC	1.73	1.82	1.64	1.89	1.84	1.91	2.02	2.58	3.10	3.13	3.20
HFP	755	1255	1656	2481	3081	3775	4141	4141	4351	4394	4544
FLI	1.26	1.31	.98	1.00	.98	1.00	1.03	1.00	1.02	1.05	1.05
FUI	1.23	1.25	.95	.99	.99	1.02	1.06	1.03	1.05	1.08	1.07
PLI	.92	.93	.93	1.25	1.43	1.64	1.90	2.69	3.05	3.14	3.21
PUI	1.01	1.16	1.26	1.86	2.18	2.54	2.94	4.06	4.97	5.13	5.24

Fig. 2. Relative Performance Comparison of Algorithms

LAPACK routines (four if you count dual cases of uplo='L' and 'U'). Performance of SRPA routines on HFP format is slightly better than LAPACK full routines while using half the storage and performance is roughly one to seven times faster than LAPACK packed routines while using the same storage.

References

[1999] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, Jack J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, D. Sorensen: LAPACK Users' Guide 3rd Ed., Society for Industrial and Applied Mathematics, pub.

[2004] J. A. Gunnels and F. G. Gustavson. *HFP Format Saves Storage and Increases Performance of LAPACK Symmetric and Triangular Packed and Full Routines*, Poster Presentation at SIAM, San Francisco, CA. Feb. 26, 2004.

A Family of High-Performance Matrix Multiplication Algorithms

John A. Gunnels[1], Fred G. Gustavson[1], Greg M. Henry[2], and Robert A. van de Geijn[3]

[1] IBM T.J. Watson Research Center
[2] Intel Corporation
[3] The University of Texas at Austin

Abstract. We describe a model of hierarchical memories and we use it to determine an optimal strategy for blocking operand matrices of matrix multiplication. The model is an extension of an earlier related model by three of the authors. As before the model predicts the form of current, state-of-the-art L1 kernels. Additionally, it shows that current L1 kernels can continue to produce their high performance on operand matrices that are as large as the L2 cache. For a hierarchical memory with L memory levels (main memory and L-1 caches), our model reduces the number of potential matrix multiply algorithms from 6^L to four. We use the shape of the matrix input operands to select one of our four algorithms. Previously four was 2^L and the model was independent of the matrix operand shapes. Because of space limitations, we do not include performance results.

1 Introduction

In this paper, we discuss an approach to implementing matrix multiplication, $C = AB + C$, that is based on a model of moving data between adjacent memory layers. This model allows us to make the following theoretical contributions: (1) At each level of the memory hierarchy most of that layer should be filled with a submatrix of one of the three operands while smaller submatrices of the other two operands are “streamed in” from the previous (slower) memory layer; (2) If a given layer is mostly filled with a submatrix of an operand, the next (faster) layer should be filled mostly with a submatrix of one of the other two operands. (3) At the L1 level, where all computations must take place, the model accurately predicts the form of two kernel routines. Embodied in points (1-3) is the general idea of using cache blocking at every memory level. The amount of work (FLOPS) done at every level is $2MNK$ on operands of size $MN + MK + KN$. We are assuming that any computer architecture, at every memory level, can feed (stream) the operands through the memory hierarchy in time less than or equal to the time for doing $2MNK$ FLOPS at maximum floating point rate (peak MFLOPs). This paper is a condensation of a 26 page paper that was written in Februrary 2002. It extends the model of [2001]. There are two new features: A new concept called (1) conservation of matrix operand sizes is used to (2) reduce the number of algorithms from 2^L to just four.

Over the last two decades, numerical linear algebra libraries have been re-designed to accommodate multi-level memory hierarchies, thereby replacing earlier libraries such as LINPACK and EISPACK. It was demonstrated, e.g. [1986,1987], that on cache-based (multi-level memory) machines, block-partitioned matrix algorithms could evince dramatic performance increases, when contrasted with their non-blocked counterparts. The key insight of block partitioning research was that it allowed the cost of $O(n^2)$ data movement between memory layers to be amortized over $O(n^3)$ computations, which had to be performed by kernel routines executing data residing in the L1 (lowest level) cache. At the base of our algorithms are *kernel* routines used in concert with submatrix partitioning (referred to as *cache blocking*). Our new memory model provides a mathematical "prediction" for the necessity of both. Prior work [1994,1997,1998], produced kernel routines, first by hand and, later, via a code generator, which employed parameter variations and careful timing in order to complete the generate-and-test cycle. The principles used in these automatic code generating versions were, for the most part, the same as those used by the ESSL library. The PHiPAC [1997] and ATLAS [1998] projects each employ a code generator approach. This methodology uses a program that writes a series of programs, each differing by a set of parameters varied by the "parent" (code writing) program. Each of these produced programs is automatically compiled and carefully timed. Using the timing characteristics of these programs, the most time-efficient code is selected. The work presented here generalizes the ATLAS model. In Section 4.1 a fuller comparison of ATLAS and our model is given. It is also related to [2002] which is discussed in Section 4.2

Our model partitions matrices A, B, and C into submatrices A_{ip}, B_{pj}, and C_{ij} and performs submatrix multiplication and this is illustrated in Section 2. In Section 3, we state a result on the minimal amount of times any matrix algorithm must bring matrix elements into and out of a given cache level. Our model achieves this lower bound complexity to within a constant factor [1999]. An important point is that A, B, and C must come through the caches multiple times. With this result in mind, Sections 2.3, 2.4 and Section 3.1 takes the L algorithmic pieces developed in Section 2 and combines them into just four different, composite algorithms. These three Sections along with Section 2.3 show why we can reduce the number of different algorithms from 2^L to just four.

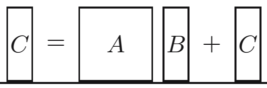
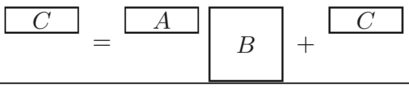
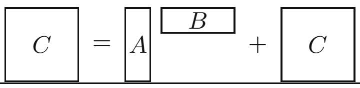
Now, since the submatrices of A, B, and C must be brought through the memory hierarchy multiple times, it pays to reformat them *once* so that data re-arrangement is not done multiple times. This also aids the effort for achieving optimal L1 kernel performance. In Sections 4, 4.1 and 4.2 we briefly describe how this is done in greater detail. Our main point is that our model features loading and storing submatrices at each cache level. Clearly, loading and storing is overhead and only serves to reduce the MFLOPS value predicted by our model. By using the new data structures, `NDS`, [2001], we can dramatically reduce the loading and storing overhead cost, thereby increasing our MFLOP rate. Because of space limitations using `NDS` in the context of our model as well as our two kernel routines that feature streaming cannot be covered in sufficient detail.

Our model does not involve automatic blocking via recursion. Recently, this area has received a great deal of attention from many important computations such as matrix factorizations, FFT, as well as matrix multiplication. See [2004]. Some researchers have referred to such methods as cache oblivious [1999]. Others have focused on "recursion" to produce new data formats for matrices, instead of the traditional Fortran and C data structures. Our view is that recursion is very powerful and excellent results are obtainable, but *only* if one combines recursion with blocking. However, whenever one neglects specific features of a computer architecture, one can expect performance to suffer.

2 Overview of Our Model

The general form of a matrix-matrix multiply is $C \leftarrow \alpha AB + \beta C$ where C is $m \times n$, A is $m \times k$, and B is $k \times n$. Table 1 introduces some terminology that we will use.

Table 1. Three basic ways to perform matrix multiply

	Condition	Shape
Matrix-panel multiply	n is small	
Panel-matrix multiply	m is small	
Panel-panel multiply	k is small	

2.1 A Cost Model for Hierarchical Memories

We will briefly model the memory hierarchy as follows:

1. The memory hierarchy consists of $L + 1$ levels, indexed $0, \ldots, L$. Level 0 corresponds to the registers, level L as main memory and the reamining levels as caches. We will often denote the hth level by L_h.
2. If $m_h \times n_h$ matrix $C^{(h)}$, $m_h \times k_h$ matrix $A^{(h)}$, and $k_h \times n_h$ matrix $B^{(h)}$ are all stored in level h of the memory hierarchy then forming $C^{(h)} \leftarrow A^{(h)}B^{(h)} + C^{(h)}$ costs time $2m_h n_h k_h \gamma_h$. γ_h is defined to be effective unit cost of a floating point operation over all $m_h n_h k_h$ multiply-adds at level L_h.

2.2 Building-Blocks for Matrix Multiplication

Consider the matrix multiplication $C^{(h+1)} \leftarrow A^{(h+1)}B^{(h+1)} + C^{(h+1)}$ where $m_{h+1} \times n_{h+1}$ matrix $C^{(h+1)}$, $m_{h+1} \times k_{h+1}$ matrix $A^{(h+1)}$, and $k_{h+1} \times n_{h+1}$ matrix $B^{(h+1)}$ are all resident in L_{h+1}. Let us assume that somehow an efficient matrix multiplication procedure exists for matrices resident in L_h. According to Table 1, we develop three distinct approaches for matrix multiplication procedures for matrices resident in L_{h+1}.

```
for $i = 1, \ldots, M_h, m_h$
    for $j = 1, \ldots, N_h, n_h$
        for $p = 1, \ldots, K_h, k_h$
            $C^{(h)}_{ij} \leftarrow A^{(h)}_{ip} B^{(h)}_{pj} + C^{(h)}_{ij}$
        endfor
    endfor
endfor
```

Fig. 1. Generic loop structure for blocked matrix-matrix multiplication

Partition

$$C^{(h+1)} = \left(\begin{array}{c|c|c} C^{(h)}_{11} & \cdots & C^{(h)}_{1N_h} \\ \hline \vdots & & \vdots \\ \hline C^{(h)}_{M_h 1} & \cdots & C^{(h)}_{M_h N_h} \end{array}\right), A^{(h+1)} = \left(\begin{array}{c|c|c} A^{(h)}_{11} & \cdots & A^{(h)}_{1K_h} \\ \hline \vdots & & \vdots \\ \hline A^{(h)}_{M^{(h)}1} & \cdots & A^{(h)}_{M^{(h)}K^{(h)}} \end{array}\right), \text{ and} \tag{1}$$

$$B^{(h+1)} = \left(\begin{array}{c|c|c} B^{(h)}_{11} & \cdots & B^{(h)}_{1N_h} \\ \hline \vdots & & \vdots \\ \hline B^{(h)}_{K_h 1} & \cdots & B^{(h)}_{K_h N_h} \end{array}\right) \tag{2}$$

where $C^{(h)}_{ij}$ is $m_h \times n_h$, $A^{(h)}_{ip}$ is $m_h \times k_h$, and $B^{(h)}_{pj}$ is $k_h \times n_h$. A blocked matrix-matrix multiplication algorithm is given in Figure 1 where the order of the loops is arbitrary. Here, M_h, N_h, and K_h are the ratios m_{h+1}/m_h, n_{h+1}/n_h, and k_{h+1}/k_h, respectively, which for clarity we assume to be integers. Further, the notation, $i = 1, \ldots, M_h, m_h$, means that the index, i iterates from 1 to M_h in steps of 1, where each block has size m_h. The unit cost γ_{h+1} will depend on the unit cost γ_h, the cost of moving data between the two memory layers, the order of the loops, the blocking sizes m_h, n_h, and k_h, and finally the matrix dimensions m_{h+1}, n_{h+1}, and k_{h+1}. The purpose of our analysis will be to determine optimal m_{h+1}, n_{h+1}, and k_{h+1} by taking into account the cost of all the above mentioned factors.

We can assume that m_h, n_h, and k_h are known when $h = 0$. Floating-point operations can only occur with data being transferred between the L1 cache (level 1) and the registers (level 0). This is, obviously, where all of the arithmetic of matrix-matrix multiply is done and it is a separate problem which is probably the most important part of any matrix-matrix multiply algorithm to get "right" (to optimize). Hence, via an induction-like argument we can, using our model, find optimal m_1, n_1, and k_1. Similar inductive reasoning leads to optimal blockings for the slower and larger memory layers $h = 2, \ldots, L$.

In the expanded version of this paper we analyze Figure 1 for all of its six variants, formed by permuting i, j, and p. For all six variants the central computation is: $C^{(h)}_{ij} \leftarrow A^{(h)}_{ip} B^{(h)}_{pj} + C^{(h)}_{ij}$. Now elements of $C^{(h)}_{ij}$, $A^{(h)}_{ip}$, and $B^{(h)}_{pj}$, are used k_h, n_h, and m_h times, respectively, during this matrix computation. Thus, we identify k_h, n_h, and m_h as the *re-use factors* of $C^{(h)}_{ij}$, $A^{(h)}_{ip}$, and $B^{(h)}_{pj}$, respectively. And, when we speak of *amortizing* the computation, we mean that elements of elements of $C^{(h)}_{ij}$, $A^{(h)}_{ip}$, and

$B_{pj}^{(h)}$, have re-use factors, k_h, n_h, and m_h and it is advantageous to make these factors as high as possible.

Also, the inner loop repetition factors K_h, N_h or M_h will each take the value one because of the aforementioned streaming resulting in " loop fusion ". The reason is given in Section 3.1 ahead. This means that the cache resident operand has an extra large re-use factor and thereby can benefit from streaming.

2.3 A Corresponding Notation

By examining Figure 1 and Table 1 the following correspondence becomes evident: $(i, j, p) \leftrightarrow$ (`RPM, RMP, RPP`). Here P stands for panel, M for matrix, and R for repeated. Think of repeated as a synomyn for streaming. For example, algorithm `RMP-RPP` is uniquely designated by its outer and inner loop indices, j and p. Similarly, algorithm `RPM-RPP` is uniquely designated by its outer and inner loop indices, i and p. For the remaining four combinations: algorithms `RPM-RMP` , `RPP-RMP` and `RMP-RPM` , `RPP-RPM` correspond to outer-inner loop pairings i, j , p, j and j, i , p, j respectively.

Floating point arithmetic must be performed out of the L1 cache. Let $h = 1$. It is clear that p should be the inner loop variable as then a `DDOT` (as opposed to a `DAXPY`) kernel can be chosen. Choosing `DDOT` saves loading and storing C values for each floating point operation. In practice, at the L_1 level, almost all implementations that we are aware of are limited to these `RMP-RPP` and `RPM-RPP` algorithms. Hence we limit ourselves to only these two algorithms at the L_1 level. The "`RPP`" is redundant and will be dropped. The `RMP`, `RPM` and `RPP` notation is fully described in [2001].

So, we have two L1 kernels, called `RMP` and `RPM`. When $h > 1$, we refer to the six algorithms illustrated in Table 1 and Figure 1 as algorithmic pieces. In the next section we will see that, for an L level memory hierarchy, $L-1$ algorithmic pieces, $h = 1, \ldots, L-1$ can be merged into a single algorithm. Because of the above-mentioned correspondence, we can label the distinct algorithms that emerge either by a string of loop indices (i, j, p) or by the analogous (`RPM, RMP, RPP`) designations. Each such string will contain $L - 1$ loop indices or $L - 1$ hyphenated symbol pairs from the (`RPM, RMP, RPP`) alphabet.

2.4 Matrix Operand Sizes Dictate the Streaming Values

Consider again Figure 1 for a particular value of h so that problem size is $m_{h+1} \times n_{h+1} \times k_{h+1}$. The outer loop index can be i, j or p. Thus we have three cases and for each there are two algorithms corresponding to the order of the middle and inner loops indices. Also, there are six possible size orderings for the problem dimensions, (e.g. $M > N > K$, etc.). It is self-evident that any ordering of the three sizes imposes an unambiguous ordering on the size of the three operands A, B, and C, involved in matrix multiplication. For example, $M > N > K$ implies that $C(M \times N)$ is larger than $A(M \times K)$, which is, in turn, larger than $B(K \times N)$. Consider the case where K is the largest dimension and $h = L - 1$, as follows:

1. $K > M > N$ which becomes $k_{h+1} > m_{h+1} > n_{h+1}$ or $|A| > |B| > |C|$.
2. $K > N > M$ which becomes $k_{h+1} > n_{h+1} > m_{h+1}$ or $|B| > |A| > |C|$.

In both cases, the algorithms used should have p as the inner loop index. Streaming provides us with the reason for this assertion. To be more specific, operand C benefits from the large streaming value of k_{h+1}. Since C is the smallest matrix in these cases, C will tend to be composed of fewer submatices making up all of C. In other words, M_h and N_h, the repetition values, will tend to be the two smallest integers, as the $m_{h+1} \times n_{h+1}$ matrix, C, is the smallest operand. Refer to case (1) above which depicts Figure 1 with loop order j, i, p. The cache resident matrix in this case is the A operand and hence N is the streaming value. In case (2) above the loop order is i, j, p and B is the cache resident matrix and M is the streaming value. Finally, in the remaining four cases we have n_{h+1} as the largest value for two cases corresponding to inner loop index j and m_{h+1} as the largest value for two cases corresponding to inner loop index i.

In Table 2 we list the six possible size-orderings for problem dimensions, M, N, and K. In cases where two or more dimensions are equal, $>$ can be viewed as $\geq$.

Table 2. The six possible matrix size orderings delineated

Case	Dimensions	Matrix Sizes	L2-Resident	L1-Resident	"Streaming" Matrix
1	$M > N > K$	$\|C\| > \|A\| > \|B\|$	A	B	C
2	$M > K > N$	$\|A\| > \|C\| > \|B\|$	C	B	A
3	$N > M > K$	$\|C\| > \|B\| > \|A\|$	B	A	C
4	$N > K > M$	$\|B\| > \|C\| > \|A\|$	C	A	B
5	$K > M > N$	$\|A\| > \|B\| > \|C\|$	B	C	A
6	$K > N > M$	$\|B\| > \|A\| > \|C\|$	A	C	B

Row entries 5 and 6 of Table 2 suggest that the C operand should be L1 cache-resident. However, for reasons given in Section 2.3, this situation leads to an inefficient inner kernel. Our solution to this dilemma is to reverse the L2/L1 residence roles of the smaller two operands.

The case $L = 2$ suggests a strategy for algorithmic choices when $L > 2$. Choose the smaller two operands to be cache-resident with the large operand supplying the streaming values. This choice is based on maximizing the streaming feature exhibited by L1 cache algorithms on current architectures and machine designs. The smaller two operands will alternate in the role of cache-resident matrix.

3 Theoretical Results on Blocking for Matrix Multiplication

Our model features block-based (submatrix) matrix multiplication. One might object to our research on the grounds that there exists some other, as yet unknown, superior method for performing the standard matrix multiply. We now state that this cannot happen as our model is optimal in the following sense:

Theorem 1. *Any algorithm that computes $a_{ik}b_{kj}$ for all $1 \leq i, j, k \leq n$ must transfer between memory and D word cache $\Omega(n^3/\sqrt{D})$ words if $D < n^2/5$.*

This theorem, was first demonstrated by Hong and Kung [1981] in 1981. Toledo [1999] gives another, simplified, proof of their result. Our model of block-based matrix multiplication transfers $O(n^3/\sqrt{D})$ words for an L1 cache of size D words. Furthermore, we can apply this theorem to every cache level residing "below" L1 in the memory pyramid: L2, L3, . . ., in the same way. Our model evinces the $\Omega(n^3/\sqrt{D_h})$ word movement, where D_h is the size of L_h, $h = 1, 2, \ldots$. To illustrate this, let us review Figure 1 and Table 1. In Figure 1 with inner loop index p, matrix C is brought into L_h once while matrices A and B are brought through L_h, N_h and M_h times, respectively. Similarly, in Figure 1 with inner loop index j, matrix A is brought into L_h once while matrices B and C are brought through L_h, M_h and K_h times, respectively. Finally, in Figure 1 with inner loop index i, matrix B is brought into L_h once while matrices A and C are brought through L_h, N_h and K_h times, respectively. Thus, we can see that the streaming matrices employed in our algorithms have the repetitive factors, N_h, M_h, and K_h.

3.1 Bridging the Memory Layers

Overall, the number of single choices is bounded by 6^{L+1} as each instance of Figure 1 has six choices and h takes on $L + 1$ values. In Section 2.3, with $h = 1$ we restricted our choice to two L1 kernels; i.e. we set the inner loop index to be p. For $h > 1$ we shall "fuse" the inner loop index at level $h + 1$ to be equal to the outer loop index at level h. This amounts to using streaming at every memory level. This means the number of algorithmic choices reduces to 2^L. Thus, Figure 1 will have only 2 loops (instead of 3) when they are combined into a composite algorithm for a processor with L caches, where cache L is memory. When L is one, we have a flat (uniform) memory and a call to either of our kernel routines, `RMP` (j_0, i_0, p_0) or `RPM` (i_0, j_0, p_0), will conform to our model. For arbitrary L, we get a nested looping structure consisting of $2(L - 1)$ loop variables, 2 each from levels 2 to L. When one considers the L1 kernel as well, there are $2L + 1$ variables (as we must add i_0, j_0, and p_0).

Furthermore, for streaming to fully work one must have matrix operands to stream from. We call this conservation of matrix operands. M, N, and K are inputs to `DGEMM`. In Table 2 and Section 2.3 we saw that were four algorithms as cases 5 and 6 were mapped to cases 2 and 4 respectively. Thus, the 2^L choices map to just four algorithms as the choice of the outer and inner indices becomes fixed for each value of h. The reason to do this is based on the conservation of matrix operand sizes for the given `DGEMM` problem. The two largest of the three input dimensions M, N, and K determine the streaming matrix as the A, B, or C operand of largest size. The four patterns that emerge for cache residency are $A, B, A, \ldots$, $B, A, B, \ldots$, $A, C, A, \ldots$, and $B, C, B, \ldots$, for $h = 1, 2, 3, \ldots$ The associated streaming values come from the two dimensions of the matrix operands C, C, B, and A respectively.

4 Practical Considerations

In the previous section we developed a model for the implementation of matrix-matrix multiplication that amortizes movement of data between memory hierarchies from a local point of view. However, there are many issues associated with actual implementation that are ignored by the analysis and the heuristic. In this section we briefly discuss some implementation details that do take some of those issues into account. We do so by noting certain machine characteristics that to our knowledge hold for a wide variety of architectures. While in the previous sections we argued from the bottom of the pyramid to the top (L_{h+1} in terms of L_h), we now start our argument at the top of the pyramid after providing some general guidelines and background.

4.1 Background and Related Work

In the Introduction, we mentioned both the PHiPAC and the ATLAS research efforts. We now describe ATLAS in more detail as it more closely relates to our contributions. In ATLAS, the code generation technique is only applied to the generation of a *single* L1 kernel. The ATLAS L1 kernel has the same form as the ESSL `DGEMM` kernel outlined in [1994] and their code generator uses many of the architectural and coding principles described in that paper. Our model predicts two types of L1 kernel and, for IBM platforms, we have an efficient implementation of each. ATLAS literature does mention this second kernel, stating that either kernel could be used and it was an arbitrary choice on their part to generate the one they selected. However, they did not pursue including this second kernel, nor did they justify their conclusion, that both kernel routines were basically the same.

Most practical matrix-matrix multiply L1 kernel routines have the form that our model predicts. For example, ESSL's kernel for the RISC-based RS/6000 processors, since their inception in 1990, have used routines that conform to this model. The same is true of the kernel for Intel's CISC Pentium III processor, which is described in an expanded version of this paper. Since ATLAS's code generator for its L1 kernel also fits our model and has shown cross-platform success, we can expect that our model will work on other platforms as well.

For the L1 level of memory, our model predicts that one should load most of the L1 cache with either the A or the B matrix operand. The other operands, C, and B or A, respectively, are streamed into and out of (through) the remainder of the L1 cache while the large A or B operand remains consistently cache-resident. Another theory predicts that each operand should be square and occupy one-third of the L1 cache. In this regard, we mention that ATLAS only uses its L1 kernel on square matrix operands. Hence the maximum operation count (multiply-add) that an invocation of the ATLAS kernel can achieve is NB^3. Our (first) model places the A operand, of size $MB \times KB$, into the L1 cache, filling most of available memory at that level. However, we can stream N, nb-sized blocks of the remaining operands through the L1 cache. By selecting nb based on the register sizes, we can allow N to be, essentially, infinite. Thus, the streamed form of our L1 kernel can potentially support $2 \times MB \times KB \times N$ flops per invocation. We observe two practical benefits from our form of kernel construction and usage:

1. A rectangular blocking where $MB < KB$ leads to a higher FLOP rate, due to the inherent asymmetry that results from having to load and store C.
2. The streaming feature allows a factor of N/NB fewer invocations of the kernel routine.

Now we turn to partitioning the matrices, A, B, and C, into comforming submatrix blocks. ATLAS's model advises only two such partitioning strategies: (1) J, I, L and (2) I, J, L, where the outer loop increments are the smallest and the inner loop, the largest. Further, for both partitionings, the ATLAS model only allows a single square blocking factor of size NB. By having only two such partitionings ATLAS documentation states that it can only block for one other memory level, for example, L2, and that their method for doing so is only approximate. Our model differs from the one employed by ATLAS in that our model has three potential blocking factors, MB, NB, and KB, at *every* cache level of the memory hierarchy.

Strangely, our model does **not** predict the two partitionings selected by ATLAS. The reason for this is that ATLAS's partitionings use K as the "streaming" parameter. In our model, the blocking parameter, KB, would be tiny. This would lead to a DAXPY-like kernel which is known to be inferior to a DDOT-like kernel because the former continually loads and stores the C operand, whereas the latter keeps the C operand in registers.

4.2 Goto BLAS

Presently the DGEMM provided by [2004] gives very high performance on a variety of platforms. The G authors think our model encompasses all the principles espoused in [2002]. TLB blocking is automatically handled when one performs data copy via the use of NDS. Our RPM B kernel is used in [2002]. And our RMP A kernel is probably not used because data copy of C is required to avoid TLB misses. The third and second last paragraphs of the Conclusion of [2002] corroborates the statements made above.

5 Summary and Conclusion

This paper extends the results of [2001] by introducing the concept of conservation of matrix operand sizes. By doing so, we show that the number of algorithms reduces from 2^L to four. It emphasizes the importance of streaming and generalizes it from L2 to L1 caches to caches $h + 1$ to h for all $h > 1$. Because of space limitations the role of NDS via data copy as well as descriptions of our two kernel routines that feature streaming is not adequately covered. Finally, our model is claimed to encompass the principles of Goto BLAS as described in [2002].

References

2001. J A Gunnels, G M Henry, and R A van de Geijn: *A Family of High-Performance Matrix Multiplication Algorithms.*, LNCS 2073, pp. 51-60. V N Alexandrov, J J Dongarra, B A Juliano, R S Renner, C J K Tan, Ed., Springer-Verlag, Pub.

1986. *ESSL Guide and Reference for IBM ES/3090 Vector Multiprocessors. Order No. SA22-7220, Feb. 1986.*, IBM Corporation.

1987. Kyle Gallivan, William Jalby, Ulrike Meier, and Ahmed Sameh, *The Impact of Hierarchical Memory Systems on Linear Algebra Algorithm Design*, CSRD Tech Report 625, University of Illinois at Urbana Champaign, pub.

1994. R. C. Agarwal, F. Gustavson, and M. Zubair, *Exploiting functional parallelism on Power2 to design high-performance numerical algorithms*, IBM Journal of Research and Development, Volume 38, Number 5, pp. 563-576, 1994.

1997. Jeff Bilmes, Krste Asanovic, Chee-whye Chin, and Jim Demmel, *Optimizing Matrix Multiply using PHiPAC: a Portable, High-Performance, ANSI C Coding Methodology*, Proc. of Int. Conf. on Supercomputing, Vienna, Austrian. July 1997.

1998. R. Clint Whaley and Jack J. Dongarra, *Automatically Tuned Linear Algebra Software*, In Proceedings of Supercomputing 1998.

2002. K. Goto and R. vandeGeijn, *On reducing TLB misses in matrix multiplication*, University of Texas at Austin, FLAME Working Note #9, November, 2002.

2001. Fred G. Gustavson, *New Generalized Matrix Data Structures Lead to a Variety of High-Performance Algorithms*, The Architecture of Scientific Software, Ronald F. Boisvert and Ping Tak Peter Tang, Ed., Kluwer Academic Press, Pub., 2001.

2004. Erik Elmroth, Fred Gustavson, Isak Jonsson, Bo Kagstrom, *Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software*, SIAM Review, Volume 46 Number 1, pp. 3-45. 2004.

1999. Matteo Frigo, Charles E. Leiserson, Harald Prokop, Sridhar Ramachandran, *Cache-Oblivious Algorithms*, Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999,IEEE Computer Society, pub.

1981. J. Hong and H. Kung, *Complexity: The Red-Blue Pebble Game*, In Proceedings of the 13th Annual ACM Symposium on Theory of Computing, pp. 326-333, 1981.

1999. Sivan Toledo, *A Survey of Out-of-Core Algorithms in Numerical Linear Algebra*, in *External Memory Algorithms and Visualization*,J Abello & J S Vitter, Ed., DIMACS Series in Disc. Math. & Theo. Comp. Sci., pp. 161-180, AMS Press, pub.

2004. Kazushige Goto, http://www.cs.utexas.edu/users/kgoto

Substructuring, Dimension Reduction and Applications: An Introduction

Organizers: Zhaojun Bai[1] and Ren-Cang Li[2]

[1] Department of Computer Science and Department of Mathematics, University of California
Davis, CA 95616, USA
bai@cs.ucdavis.edu

[2] Department of Mathematics, University of Kentucky, Lexington, KY 40506, USA
rcli@ms.uky.edu

There are a variety of reasons to go for substructuring and dimension reduction in scientific computations and applications. Substructuring makes it possible to solve large and seemingly intractable computational problems solvable in today technology by some kind of Divide-and-Conquer technique; Substructuring offers a general methodology to do parallelization; And substructuring allows one to design algorithms to preserve substructures at a very fine level of underlying problems of interest, which usually go unnoticed by more general purposed methods. Often if done right, payoff will be significant. Dimension reduction is a rather broader concept referring to techniques that achieve significant reductions of problem sizes so as to make intractable numerical simulations tractable. Successful examples are abundant, including reduced order modelling from dynamical systems and circuit design, cluster text data analysis, and data mining. This minisymposium presents currently active researches in substructuring strategies, model reduction and applications, and domain decompositions, among others.

Parallel Algorithms for Balanced Truncation Model Reduction of Sparse Systems[⋆]

José M. Badía[1], Peter Benner[2], Rafael Mayo[1], and Enrique S. Quintana-Ortí[1]

[1] Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I
12.071–Castellón, Spain
{badia,mayo,quintana}@icc.uji.es

[2] Fakultät für Mathematik, Technische Universität Chemnitz
D-09107 Chemnitz, Germany
benner@mathematik.tu-chemnitz.de

Abstract. We describe the parallelization of an efficient algorithm for balanced truncation that allows to reduce models with state-space dimension up to $\mathcal{O}(10^5)$. The major computational task in this approach is the solution of two large-scale sparse Lyapunov equations, performed via a coupled LR-ADI iteration with (super-)linear convergence. Experimental results on a cluster of Intel Xeon processors illustrate the efficacy of our parallel model reduction algorithm.

1 Introduction

Consider the continuous linear time-invariant (LTI) dynamical system in state-space form:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t), \quad t > 0, \quad x(0) = x^0, \\ y(t) &= Cx(t) + Du(t), \quad t \geq 0, \end{aligned} \tag{1.1}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, $D \in \mathbb{R}^{p\times m}$, and $x^0 \in \mathbb{R}^n$ is the initial state of the system. Also, the number of states, n, is known as the state-space dimension (or the order) of the system, and the associated transfer function matrix (TFM) is defined by $G(s) = C(sI - A)^{-1}B + D$. Hereafter, we assume that the spectrum of the state matrix, A, is contained in the open left half plane, implying that the system is stable.

The model reduction problem requires finding a reduced-order LTI system,

$$\begin{aligned} \dot{\hat{x}}(t) &= \hat{A}\hat{x}(t) + \hat{B}\hat{u}(t), \quad t > 0, \quad \hat{x}(0) = \hat{x}^0, \\ \hat{y}(t) &= \hat{C}\hat{x}(t) + \hat{D}\hat{u}(t), \quad t \geq 0, \end{aligned} \tag{1.2}$$

of order r, $r \ll n$, and associated TFM $\hat{G}(s) = \hat{C}(sI - \hat{A})^{-1}\hat{B} + \hat{D}$ which approximates $G(s)$. The reduced-order model can then replace the original high-order system in subsequent calculations including, e.g., the design of controllers [19], thus saving valuable hardware resources.

[⋆] José M. Badía, Rafael Mayo, and E.S. Quintana-Ortí were supported by the CICYT project No. TIC2002-004400-C03-01 and FEDER, and project No. P1B-2004-6 of the *Fundación Caixa-Castellón/Bancaixa and UJI*. P. Benner was supported by the DFG Research Center "Mathematics for key technologies" (FZT 86) in Berlin.

Model reduction of moderately large linear systems (n in the hundreds) arises, e.g., in control of large flexible mechanical structures or large power systems [5,7,8,11,20]. Problems of such dimension can be reduced using libraries such as SLICOT[3] or the MATLAB control-related toolboxes on current desktop computers. Larger problems can still be reduced by using the methods in PLiCMR[4] on parallel computers [3,4]. Nevertheless, the applicability of these methods is ultimately limited as they do not exploit the usual sparse structure of the state matrix A and thus present a computational cost of $\mathcal{O}(n^3)$ floating-point arithmetic operations (flops) and require storage for $\mathcal{O}(n^2)$ real numbers.

Therefore, a different approach is necessary for very large sparse systems, with state-space dimension as high as $\mathcal{O}(10^5)$–$\mathcal{O}(10^6)$, such as those arising, among others, in weather forecast, circuit simulation and VLSI design, as well as air quality simulation (see, e.g., [2,6,13,14]). The model reduction method considered in this paper is based on the so-called state-space truncation and requires, at a first stage, the solution of two large sparse Lyapunov equations whose coefficient matrix is the state matrix of the system. A low-rank iteration [17,22] is employed for this purpose which only involves sparse numerical computations such as the solution of linear systems and matrix-vector products. The reduced-order system is then obtained using a slightly modified version of the balanced truncation (BT) method [4,21], and only requires dense linear algebra operations on much smaller data matrices.

Although there exist several other approaches for model reduction (see [2,12] and the references therein), those are specialized for certain problem classes and often lack properties such as error bounds or preservation of stability, passivity, or phase information. A complete comparison between the numerical properties of SVD-based methods (as BT) and Krylov subspace methods can be found in [2].

The paper is structured as follows. In Section 2 we briefly review the method for model reduction of sparse linear systems, based on the BT method and low-rank solvers for the Lyapunov equations arising in state-space truncation. In Section 3 we describe a few parallelization and implementation details of our model reduction algorithm. Finally, the efficacy of the algorithm is reported in Section 4, and some concluding remarks follow in Section 5.

2 Model Reduction of Sparse Linear Systems

2.1 The Square-Root BT Method

BT model reduction [18,23,24,25] belongs to the family of absolute error methods which aim at minimizing $\|\Delta_a\|_\infty = \|G - \hat{G}\|_\infty$. Here $\|G\|_\infty$ denotes the $\mathcal{L}_\infty$- or $\mathcal{H}_\infty$-norm of a stable, rational matrix function defined as

$$\|G\|_\infty = \sup_{\omega \in \mathbb{R}} \sigma_{\max}(G(\jmath\omega)), \tag{2.3}$$

where $\jmath := \sqrt{-1}$ and $\sigma_{\max}(M)$ stands for the largest singular value of M.

[3] Available from `http://www.slicot.net`

[4] Available from `http://spine.act.uji.es/~plicmr.html`

BT methods are strongly related to the controllability Gramian W_c and the observability Gramian W_o of the system 1.1, given by the solutions of the two dual Lyapunov matrix equations:

$$AW_c + W_cA^T + BB^T = 0, \quad A^TW_o + W_oA + C^TC = 0. \tag{2.4}$$

As the state matrix A is assumed to be stable, W_c and W_o are positive semidefinite and therefore can be factored as $W_c = S^TS$ and $W_o = R^TR$, where S and R are the corresponding *Cholesky factors*.

Efficient Lyapunov solvers that exploit the sparsity of the coefficient matrix A and provide *full-rank factors* $\hat{S}$ and $\hat{R}$ that can replace S and R in subsequent computations are described in the next subsection.

Consider now the singular value decomposition (SVD) of the product

$$SR^T = U\Sigma V^T = [\,U_1\ U_2\,] \begin{bmatrix} \Sigma_1 & \\ & \Sigma_2 \end{bmatrix} [\,V_1\ V_2\,]^T, \tag{2.5}$$

where U and V are orthogonal matrices, and $\Sigma = \mathrm{diag}\,(\sigma_1, \sigma_2, \ldots, \sigma_n)$ is a diagonal matrix containing the singular values $\sigma_1, \ldots, \sigma_n$ of SR^T, which are called the *Hankel singular values* of the system.

Both common versions of BT, the *square-root* (SR) and *balancing-free square-root* (BFSR) BT algorithms, allow an adaptive choice of the state-space dimension r of the reduced-order model as they provide a realization $\hat{G}$ which satisfies

$$\|\Delta_a\|_\infty = \|G - \hat{G}\|_\infty \leq 2 \sum_{j=r+1}^{n} \sigma_j. \tag{2.6}$$

Besides, if $\sigma_r > \sigma_{r+1} = 0$, then r is the state-space dimension of a minimal realization of the system.

SR BT methods determine the reduced-order model of order r as

$$\hat{A} = T_lAT_r, \quad \hat{B} = T_lB, \quad \hat{C} = CT_r, \quad \hat{D} = D, \tag{2.7}$$

with the projection matrices T_l and T_r given by

$$T_l = \Sigma_1^{-1/2}V_1^TR \quad \text{and} \quad T_r = S^TU_1\Sigma_1^{-1/2}. \tag{2.8}$$

BFSR BT algorithms often provide more accurate reduced-order models in the presence of rounding errors.These algorithms obtain T_l and T_r from the following two QR factorizations,

$$S^TU_1 = [P_1\ P_2] \begin{bmatrix} R_s \\ 0 \end{bmatrix}, \qquad R^TV_1 = [Q_1\ Q_2] \begin{bmatrix} R_r \\ 0 \end{bmatrix}, \tag{2.9}$$

where the columns of $P_1,\ Q_1 \in \mathbb{R}^{n\times r}$ form orthonormal bases for $\mathrm{range}(T_r)$, $\mathrm{range}(T_l)$, respectively, and $R_s,\ R_r \in \mathbb{R}^{r\times r}$ are upper triangular. The reduced system is then given by 2.7 and the projection matrices

$$T_l = (Q_1^TP_1)^{-1}Q_1^T, \qquad T_r = P_1. \tag{2.10}$$

2.2 Low Rank Solution of Lyapunov Equations

In this subsection we revisit the Lyapunov solvers introduced in [17,22]. These iterative algorithms benefit from the frequently encountered low-rank property of the constant terms in 2.4 to provide low-rank approximations to a Cholesky or full-rank factor of the solution matrix. These approximations can reliably substitute S and R in the computation of 2.5, 2.8, and 2.9.

Specifically, given an "l–cyclic" set of (complex) shift parameters $\{p_1, p_2, \ldots\}$, $p_k = \alpha_k + \beta_k\, \jmath$, such that $p_k = p_{k+l}$, the cyclic *low-rank alternating direction implicit* (LR-ADI) iteration proposed in [22] can be reformulated as follows:

$$\begin{aligned} V_0 &= (A + p_1 I_n)^{-1} B, & \hat{S}_0 &= \sqrt{-2\,\alpha_1}\, V_0, \\ V_{k+1} &= V_k - \delta_k (A + p_{k+1} I_n)^{-1} V_k, & \hat{S}_{k+1} &= \left[\hat{S}_k \;,\; \gamma_k V_{k+1} \right], \end{aligned} \tag{2.11}$$

where $\gamma_k = \sqrt{\alpha_{k+1}/\alpha_k}$, $\delta_k = p_{k+1} + \overline{p_k}$, $\overline{p_k}$ being the conjugate of p_k, and I_n denotes the identity matrix of order n. On convergence, after $k_{\max}$ iterations, a low-rank matrix $\hat{S}_k$ of order $n \times k_{\max} m$ is computed such that $\hat{S}_k \hat{S}_k^T$ approximates $W_c = S^T S$. An analogous iteration involving A^T provides a low-rank approximation $\hat{R}_k$ of R.

The performance of iteration 2.11 strongly depends on the selection of the shift parameters. In practice, the set $\{p_1, p_2, \ldots, p_l\}$ should be chosen so that it is closed under complex conjugation. In case the eigenvalues of A are real, the optimal parameters can be computed explicitly [27]. Otherwise, an optimal solution is not known. A heuristic procedure is proposed in [22] to compute the parameters by approximating the eigenvalues of A and A^{-1} of largest magnitude. The procedure is based on an Arnoldi iteration involving A and A^{-1}. For further details on the convergence of the LR-ADI iteration and the properties of the heuristic selection procedure, see [22].

It should be emphasized that the methods just described for solving 2.4 and 2.5 significantly differ from standard methods used in the MATLAB toolboxes or SLICOT [26]. First, the proposed LR-ADI iteration for the solution of the dual Lyapunov equation exploits the sparsity in the coefficient matrix. Besides, as we are using low-rank approximations to the full-rank or Cholesky factors, the computation of the SVD in 2.5 is usually much more efficient: instead of a computational cost of $\mathcal{O}(n^3)$ when using the Cholesky factors, this approach leads to an $\mathcal{O}(k_{\max}^2 \cdot m \cdot p \cdot n)$ cost where, in model reduction, often $m, p \ll n$; see [4]. This latter advantage is shared by the routines in our dense parallel model reduction library PLiCMR [4]. However, PLiCMR routines do not exploit the sparsity of the coefficient matrix.

3 Parallel Implementation

3.1 Implementation Details

Iteration 2.11 can be stopped when the contribution of the columns added to $\hat{S}_{k+1}$ is negligible; thus, in practice, we stop the iteration when $||\gamma_k V_{k+1}||_1$ is "small".

The use of direct linear system solvers [10] (based, e.g., on the LU or Cholesky factorization) is appealing as the same coefficient matrix is involved in iterations k

and $k + l$. Therefore, the computed factorization can be re-used several times provided sufficient workspace is available to store the factors. Also, many direct sparse solvers that include an initial phase of symbolic factorization only need to perform this phase once, as all linear systems share the same sparsity pattern.

Notice that even in case all data matrices are real, the LR-ADI iteration will involve complex arithmetic in case any of the shifts is complex. Special care must be taken so that all shifts are real in case all eigenvalues of A are real as that reduces the computational cost of the iteration significantly.

3.2 Parallelization

The LR-ADI iteration and the computation of the acceleration shifts basically require linear algebra operations such as matrix-vector products (in the Arnoldi procedure for computation of the shifts for the iteration) and the solution of linear systems (also in the computation of the shifts, and in the LR-ADI iteration). Our approach for dealing with these matrix operations is based on the use of parallel linear algebra libraries. (For an extensive list of those, visit

`http://www.netlib.org/utk/people/JackDongarra/la-sw.html`.)

Specifically, in case sparse matrices are involved, the matrix-vector products are computed as a series of "usaxpy" ($y = y + \alpha \cdot x$) operations, with each process in the parallel system performing, approximately, an equal amount of these operations. Although beneficial, no attempt to balance the computational load is done here as the contribution of the matrix-vector products to the cost of the model reduction algorithm is a minor one: in practice, the number of matrix-vector products is proportional to the number of different shifts, and much smaller than the number of linear systems that need to be solved (one per iteration). Besides, the cost of a matrix-vector product is in general much lower than that of solving a linear system.

Sparse linear systems can be solved in our parallel codes by using appropriate kernels from MUMPS or SuperLU [1,9]. In both cases, the analysis phase (corresponding to a symbolic factorization) is performed only once for the iteration, and the factorization phase is only performed during the first l iterations.

In case the coefficient matrix is banded we use the linear system solvers in ScaLAPACK. We also implemented our own parallel routine for the banded matrix-vector product as this is not available in the library.

Our parallel codes can also deal with dense linear systems using the kernels in ScaLAPACK. However, in such case the methods in PLiCMR (see [4] for an overview) are preferred as the Lyapunov equation solvers used there have quadratic convergence rate as opposed to the (super-)linear convergence of the ADI method at a comparable cost of the remaining computational stages.

After the LR-ADI iteration is completed, the model reduction algorithm proceeds to compute the product of $\hat{S}_k \hat{R}_k^T$ as in 2.5 and the reduced-order model using 2.8 or 2.9–2.10. As both $\hat{S}_k$ and $\hat{R}_k$ are full dense factors, their product and the SVD are computed using the kernels in ScaLAPACK. The SVD of a matrix with complex entries is not available in ScaLAPACK. This difficulty is circumvented by collecting the product

$\hat{S}_k \hat{R}_k^T$ into a single node and computing its SVD using the serial routine from LAPACK. We exploit here that the size of the resulting matrix is order of $k_{\max} m \times k_{\max} p$ which, in general, is much smaller than $n \times n$.

Once the projection matrices T_l and T_r have been computed, it only remains to apply these matrices to form the reduced-order system $(T_l A T_r, T_l B, C T_r, D)$. These matrix products are performed by taking into account the characteristics of matrices A, B, and C which can be sparse, banded, or dense. In the former two cases, the product is computed as a series of matrix-vector products, while in the latter case we rely again on the corresponding ScaLAPACK routine.

4 Experimental Results

All the experiments presented in this section were performed on a cluster of $n_p = 32$ nodes using IEEE double-precision floating-point arithmetic ($\varepsilon \approx 2.2204 \times 10^{-16}$). Each node consists of an Intel Xeon processor at 2.4 GHz with 1 GByte of RAM. We employ a BLAS library specially tuned for this processor that achieves around 3800 Mflops (millions of flops per second) for the matrix product (routine DGEMM) [15]. The nodes are connected via a *Myrinet* multistage network and the MPI communication library is specially developed and tuned for this network. The performance of the interconnection network was measured by a simple loop-back message transfer resulting in a latency of 18 μsec. and a bandwidth of 1.4 Gbit/sec.

Our benchmark for model reduction of sparse systems consists of three scalable models and serves to illustrate the benefits gained from applying model reduction via BT combined with parallelism to large-scale systems. This is by no means a comparison of the efficacy of the direct linear system solvers in MUMPS, SuperLU, and ScaLAPACK. In order to avoid it, we report results using a different solver and problem size for each example.

As in all our experiments both SR and BFSR BT algorithms delivered closely similar results, we only report data for the first algorithm.

Example 1. This example models the heat diffusion in a (1-dimensional) thin rod with a single heat source [5]. The system is parameterized by a scalar α that we set to $\alpha = 0.1$ in our experiments. The spatial domain is discretized into segments of length $h = \frac{1}{n+1}$ and centered differences are used to approximate the diffusion operator. A heat point is assumed to be located at 1/3 of the length of the rod and the temperature is recorded at 2/3 of the length.

In this case, a model of state-space dimension $n = 10,000$ was reduced to order $r = 85$ using the sparse direct solvers in SuperLU 2.0 (distr.) and $l = 50$ different shifts for the LR-ADI iteration. Only 16 nodes of the parallel system were used resulting in the following execution times (expressed in minutes and seconds) for each one of the stages of the algorithm:

Comp. shifts	LR-ADI iter.	Comp. SVD+SR	Total
11.74” (11.3%)	1’ 28” (86.1%)	2.49” (2.4%)	1’ 42”

Thus, the computation of the reduced-order model required slightly more than 1.5 minutes. This is clearly a low temporal cost that we may be willing to pay for the great benefits obtained from using the reduced-order model in the subsequent computations needed for simulation, optimization, or control purposes.

These results also show that the most expensive part of the model reduction procedure was the LR-ADI iteration, which required about 86% of the total time. Notice that, as $l = 50$, this phase requires the computation of 50 matrix (LU) factorizations, while the computation of the shifts only requires one factorization. Convergence was achieved for this example after 128 iterations producing full-rank approximations of the Cholesky factors, $\hat{S}_k$ and $\hat{R}_k$, of order $128 \times n$. Thus, the SVD computation only involved a square matrix of order 128 requiring around 2.5 sec.

The reduced-order system obtained with our method satisfies the absolute error bound $\|G - \hat{G}\|_\infty \approx 1.91 \times 10^{-16}$, showing that it is possible to reduce the order of the system from $n = 10,000$ to 85 without a significative difference between the behavior of the original and the reduced-order models. Actually, the reduced-order model can be seen as a numerically minimal realization of the LTI system.

Example 2. The matrices in this case model the temperature distribution in a 2-D surface. The state matrix of the system presents a block tridiagonal sparse structure of order N^2 and is obtained from a discretization of the Poisson's equation with the 5-point operator on an $N \times N$ mesh. The input and output matrices were constructed as $B = \left[e_1^T,\ 0_{1\times N(N-1)}\right]^T$ (with $e_1 \in \mathbb{R}^N$ the first column of the identity matrix) and $C = \left[e_1^T,\ 1,\ 0_{1\times N-2},\ 1\right]$ (with $e_1 \in \mathbb{R}^{N(N-1)}$). A large system of order $n = 202,500$, with a single input and a single output, was reduced in this case to order $r = 30$. The parallel algorithm employed MUMPS 4.3 as the direct linear system solver, $l = 15$ shifts, and 16 nodes, resulting in the following execution times:

Comp. shifts	LR-ADI iter.	Comp. SVD+SR	Total
9.62" (0.7%)	19' 54" (97.6%)	19.9" (1.6%)	20' 22"

Here, obtaining the reduced-order model required about 20 minutes. This time could be further reduced by employing more nodes of the system. In this example the most expensive part was again the LR-ADI iteration, which required the major part of the execution time of the procedure. Convergence was achieved after 69 iterations, producing a reduced-order model which satisfies $\|G - \hat{G}\|_\infty \approx 3.2 \times 10^{-17}$. From this absolute error bound we again conclude that, for this example, it is possible to reduce the order of the system from $n = 202,500$ to 30 without a significative difference between the behavior of the original and the reduced-order models.

Example 3. The matrices in this example model a RLC circuit of n_0 sections interconnected in cascade resulting in a system with $n = 2n_0$ states and a single input/output [16]. The system is parameterized by scalars $R = 0.1$, $\bar{R} = 1.0$, $C = 0.1$, and $L = 0.1$. The state matrix in this example is tridiagonal.

In this third example we reduced a system of order $n = 200,000$ and a single input and output to order $r = 50$. Using the solver in ScaLAPACK 1.6 for banded linear systems, $l = 15$ shifts for the LR-ADI iteration, and 16 nodes of the parallel system, the model reduction algorithm required the following execution times:

Comp. shifts	LR-ADI iter.	Comp. SVD+SR	Total
2.26" (7.4%)	5.14" (16.9%)	22.87" (75.4%)	30.33"

The results show that scarcely more than half a minute was sufficient to reduce a large scale system of order $n = 200,000$. Convergence of the LR-ADI iteration was achieved for this example after 57 iterations, resulting in a reduced-order system that, for $r = 50$, satisfied $\|G - \hat{G}\|_\infty < 4.9 \times 10^{-23}$.

There are two important differences between the results for Examples 2 and 3. First, although the order of the models are of the same magnitude, the execution times for Example 3 are much lower. This is due to the ability of the banded solvers in ScaLAPACK to exploit the structure of the state matrix for Example 3. Second, the largest part of the computation time lies for Example 3 in the computation of the SVD and the reduced-order system (SVD+SR stage) while, for Example 2, it is spent in the LR-ADI iteration. However, the execution times of the SVD+SR stage are comparable for both examples. Thus, it is really the important reduction of the execution time of the LR-ADI iteration which, for Example 3, produced the shift of the bulk of the computation time to the SVD+SR stage.

5 Concluding Remarks

We have presented parallel algorithms for BT model reduction of large sparse (and dense) linear systems of order as large as $\mathcal{O}(10^5)$. Our model reduction algorithms employ kernels from parallel linear algebra libraries such as SuperLU, MUMPS, and ScaLAPACK. Three large-scale models are used in the experimental results to illustrate the benefits gained from applying model reduction via BT combined with parallel execution and to report the performance of the approach on a cluster of Intel Xeon processors.

The parallelism of our approach depends on the problem structure and the parallelism of the underlying parallel linear algebra library (SuperLU, MUMPS, or the banded codes in ScaLAPACK).

References

1. P.R. Amestoy, I.S. Duff, J. Koster, and J.-Y. L'Excellent. MUMPS: a general purpose distributed memory sparse solver. In *Proc. PARA2000, 5th International Workshop on Applied Parallel Computing*, pages 122–131, 2000.
2. A.C. Antoulas. *Approximation of Large-Scale Dynamical Systems*. SIAM Publications, Philadelphia, PA, 2005.
3. P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. Balanced truncation model reduction of large-scale dense systems on parallel computers. *Math. Comput. Model. Dyn. Syst.*, 6(4):383–405, 2000.
4. P. Benner, E.S. Quintana-Ortí, and G. Quintana-Ortí. State-space truncation methods for parallel model reduction of large-scale systems. *Parallel Comput.*, 29:1701–1722, 2003.
5. Y. Chahlaoui and P. Van Dooren. A collection of benchmark examples for model reduction of linear time invariant dynamical systems. SLICOT Working Note 2002–2, February 2002. Available from `http://www.win.tue.nl/niconet/NIC2/reports.html`.
6. C.-K. Cheng, J. Lillis, S. Lin, and N.H. Chang. *Interconnect Analysis and Synthesis*. John Wiley & Sons, Inc., New York, NY, 2000.
7. J. Cheng, G. Ianculescu, C.S. Kenney, A.J. Laub, and P. M. Papadopoulos. Control-structure interaction for space station solar dynamic power module. *IEEE Control Systems*, pages 4–13, 1992.

8. P.Y. Chu, B. Wie, B. Gretz, and C. Plescia. Approach to large space structure control system design using traditional tools. *AIAA J. Guidance, Control, and Dynamics*, 13:874–880, 1990.
9. J.W. Demmel, J.R. Gilbert, and X.S. Li. *SuperLU User's Guide*.
10. I.S. Duff, A.M. Erisman, and J.K. Reid. *Direct methods for sparse matrices*. Oxford Science Publications, Oxford, UK, 1993.
11. L. Fortuna, G. Nummari, and A. Gallo. *Model Order Reduction Techniques with Applications in Electrical Engineering*. Springer-Verlag, 1992.
12. R. Freund. Reduced-order modeling techniques based on Krylov subspaces and their use in circuit simulation. In B.N. Datta, editor, *Applied and Computational Control, Signals, and Circuits*, volume 1, chapter 9, pages 435–498. Birkhäuser, Boston, MA, 1999.
13. R. Freund. Model reduction methods based on Krylov subspaces. *Acta Numerica*, 12:267–319, 2003.
14. R. W. Freund and P. Feldmann. Reduced-order modeling of large passive linear circuits by means of the SyPVL algorithm. In *Technical Digest of the 1996 IEEE/ACM International Conference on Computer-Aided Design*, pages 280–287. IEEE Computer Society Press, 1996.
15. K. Goto and R. van de Geijn. On reducing TLB misses in matrix multiplication. FLAME Working Note 9, Department of Computer Sciences, The University of Texas at Austin, `http://www.cs.utexas.edu/users/flame`, 2002.
16. S. Gugercin and A.C. Antoulas. A survey of balancing methods for model reduction. In *Proc. European Control Conf. ECC 03* (CD-ROM), Cambridge, 2003.
17. J.-R. Li and J. White. Low rank solution of Lyapunov equations. *SIAM J. Matrix Anal. Appl.*, 24(1):260–280, 2002.
18. B.C. Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Trans. Automat. Control*, AC-26:17–32, 1981.
19. G. Obinata and B.D.O. Anderson. *Model Reduction for Control System Design*. Communications and Control Engineering Series. Springer-Verlag, London, UK, 2001.
20. C.R. Paul. *Analysis of Multiconductor Transmission Lines*. Wiley–Interscience, Singapur, 1994.
21. T. Penzl. Algorithms for model reduction of large dynamical systems. Technical Report SFB393/99-40, Sonderforschungsbereich 393 *Numerische Simulation auf massiv parallelen Rechnern*, TU Chemnitz, 09107 Chemnitz, Germany, 1999. Available from `http://www.tu-chemnitz.de/sfb393/sfb99pr.html`.
22. T. Penzl. A cyclic low rank Smith method for large sparse Lyapunov equations. *SIAM J. Sci. Comput.*, 21(4):1401–1418, 2000.
23. M.G. Safonov and R.Y. Chiang. A Schur method for balanced-truncation model reduction. *IEEE Trans. Automat. Control*, AC–34:729–733, 1989.
24. M.S. Tombs and I. Postlethwaite. Truncated balanced realization of a stable non-minimal state-space system. *Internat. J. Control*, 46(4):1319–1330, 1987.
25. A. Varga. Efficient minimal realization procedure based on balancing. In *Prepr. of the IMACS Symp. on Modelling and Control of Technological Systems*, volume 2, pages 42–47, 1991.
26. A. Varga. Model reduction software in the SLICOT library. In B.N. Datta, editor, *Applied and Computational Control, Signals, and Circuits*, volume 629 of *The Kluwer International Series in Engineering and Computer Science*, pages 239–282. Kluwer Academic Publishers, Boston, MA, 2001.
27. E.L. Wachspress. ADI iteration parameters for the Sylvester equation, 2000. Available from the author.

Towards an Optimal Substructuring Method for Model Reduction

Zhaojun Bai[1] and Ben-Shan Liao[2]

[1] Department of Computer Science and Department of Mathematics
University of California at Davis
Davis, CA 95616
bai@cs.ucdavis.edu
[2] Department of Mathematics
University of California at Davis
Davis, CA 95616
liao@math.ucdavis.edu

Abstract. Substructuring methods have been studied since 1960s. The modes of subsystems associated with the lowest frequencies are typically retained. This mode selection rule is largely heuristic. In this paper, we use a moment-matching analysis tool to derive a new mode selection criterion, which is compatible to the one recently derived by Givoli *et al* using Dirichlet-to-Neumann (DtN) map as an analysis tool. The improvements of the new mode selection criterion are demonstrated by numerical examples from structural dynamics and MEMS simulation.

1 Introduction

Model-order reduction techniques play an indispensable role to meet the continual and compelling need for accurately and efficiently simulating dynamical behavior of large and complex physical systems. One popular method is the substructuring or the component mode synthesis (CMS), which was developed back to early 1960s [7,8,4]. CMS explicitly exploits underlying structures of a system and effectively avoids the expenses of processing the entire system at once. The model-order reduction of subsystems can be conducted in parallel. The subsystem structure is preserved.

Specifically, in this paper, we consider a lumped MIMO dynamical system of the form

$$\Sigma_N : \begin{cases} M\ddot{x}(t) + Kx(t) = Bu(t), \\ \qquad\qquad y(t) = L^T x(t), \end{cases} \tag{1.1}$$

with the initial conditions $x(0) = x_0$ and $\dot{x}(0) = v_0$. Here t is the time variable, $x(t) \in \mathcal{R}^N$ is a state vector, N is the degree of freedoms (DOFs), $u(t) \in \mathcal{R}^p$ the input excitation force vector, and $y(t) \in \mathcal{R}^m$ the output measurement vector. $B \in \mathcal{R}^{N\times p}$ and $L \in \mathcal{R}^{N\times m}$ are input and output distribution arrays, respectively. M and K are system matrices, such as mass and stiffness. Assume that M is symmetric positive definite and K is symmetric semidefinite. Furthermore, the state vector $x(t)$ and the system matrices M and K are posed of subsystem structure, namely, they are partitioned into the three blocks, representing subsystems I, II and interface:

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix}, \; M = \begin{bmatrix} M_{11} & & M_{13} \\ & M_{22} & M_{23} \\ M_{13}^T & M_{23}^T & M_{33} \end{bmatrix}, \; K = \begin{bmatrix} K_{11} & & K_{13} \\ & K_{22} & K_{23} \\ K_{13}^T & K_{23}^T & K_{33} \end{bmatrix}. \tag{1.2}$$

We denote the number of DOFs of subsystems I, II and the interface by N_1, N_2 and N_3, respectively. Thus the total number of DOFs of Σ_N is $N = N_1 + N_2 + N_3$.

By Laplace transform, the input-output behavior of Σ_N in the frequency domain is characterized by the transfer function

$$H(\omega) = L^T(-\omega^2 M + K)^{-1} B,$$

where ω is referred to as the frequency. For the simplicity of exposition, we have assumed that $x(0) = \dot{x}(0) = 0$.

A substructuring method replaces the system Σ_N with a system of the same form but (much) smaller dimension of the state-vector $z(t)$:

$$\Sigma_n : \begin{cases} M_n \ddot{z}(t) + K_n z(t) = B_n \, \mathbf{u}(t), \\ \qquad\qquad\quad \widehat{y}(t) = L_n^T \, z(t), \end{cases} \tag{1.3}$$

such that the input-output behavior of Σ_n is an acceptable approximation of Σ_N. The number of DOFs of the new state-vector $z(t)$ is $n = n_1 + n_2 + N_3$ with $n_1 < N_1$ and $n_2 < N_2$. The DOFs of the interface block is unchanged. Furthermore, M_n and K_n preserve the block structures of M and K.

A key step in substructuring methods is to compute and retain the modes of subsystems. A standard mode selection practice is to retain the modes associated with few lowest frequencies. This is largely heuristic and does not guarantee to produce an optimal reduced system Σ_n as shown by the following simple example. Let

$$M = \left[\begin{array}{c|cc|c} 1 & & & 0.7 \\ \hline & \mathbf{1} & & 10^{-3} \\ & & \mathbf{1} & 0.3 \\ \hline 0.7 & 10^{-3} & 0.3 & 1 \end{array}\right], \; K = \left[\begin{array}{c|cc|c} 0.9 & & & \\ \hline & \mathbf{1} & & \\ & & \mathbf{2} & \\ \hline & & & 1 \end{array}\right], \; B = \left[\begin{array}{c} 1 \\ \hline 0 \\ 0 \\ \hline 0 \end{array}\right], \; L = \left[\begin{array}{c} 1 \\ \hline 1 \\ 1 \\ \hline 1 \end{array}\right]. \tag{1.4}$$

Suppose the subsystem II is reduced. Then by the standard lowest frequency mode selection criterion, the reduced system Σ_n is given by

$$M_n = \left[\begin{array}{c|c|c} 1 & & 0.7 \\ \hline & \mathbf{1} & 10^{-3} \\ \hline 0.7 & 10^{-3} & 1 \end{array}\right], \; K_n = \left[\begin{array}{c|c|c} 0.9 & & \\ \hline & \mathbf{1} & \\ \hline & & 1 \end{array}\right], \; B_n = \left[\begin{array}{c} 1 \\ \hline 0 \\ \hline 0 \end{array}\right], \; L_n = \left[\begin{array}{c} 1 \\ \hline 1 \\ \hline 1 \end{array}\right]. \tag{1.5}$$

However, if we retain the other mode in the system II, then the reduced system $\widehat{\Sigma}_n$ is given by

$$\widehat{M}_n = \left[\begin{array}{c|c|c} 1 & & 0.7 \\ \hline & \mathbf{1} & 0.3 \\ \hline 0.7 & 0.3 & 1 \end{array}\right], \; \widehat{K}_n = \left[\begin{array}{c|c|c} 0.9 & & \\ \hline & \mathbf{2} & \\ \hline & & 1 \end{array}\right], \; \widehat{B}_n = \left[\begin{array}{c} 1 \\ \hline 0 \\ \hline 0 \end{array}\right], \; \widehat{L}_n = \left[\begin{array}{c} 1 \\ \hline 1 \\ \hline 1 \end{array}\right]. \tag{1.6}$$

Figure 1 shows the magnitudes (in log of base 10) of the transfer function $H(\omega)$ of the original system Σ_N and the reduced ones $H_n(\omega)$ (called CMS line) and $\widehat{H}_n(\omega)$ (called CMS$_\chi$ line). It is clear that the low-frequency dominant mode selection criterion is not optimal.

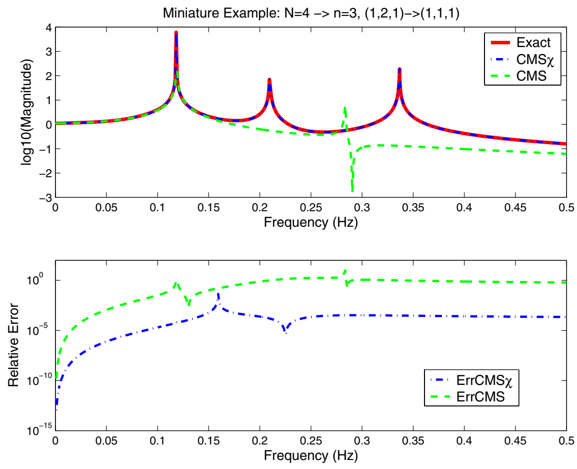


Fig. 1. The frequency response analysis (top) and relative error (bottom) for the miniature example

A question that arises naturally is "*which are the important modes of subsystems?*" In the recent work of Givoli *et al* [1,6], an optimal modal reduction (OMR) algorithm is proposed. In contrast to the low-frequency dominant mode selection rule, they introduce the concept of *coupling matrix*-based mode selection criterion. The concept is derived via the DtN map analysis tool, originally developed for solving partial differential equations with non-reflecting boundary conditions [9]. They show that the OMR method is better than the standard modal reduction (SMR) method. However, there are a number of limitations in the OMR method, such as the assumption of external force $Bu(t)$ only applied to one of the subsystems.

In this paper, we present an alternative mode selection criterion to the CMS method. The resulting method is called CMS_χ. The new mode selection criterion in CMS_χ is derived in an algebraic setting based on the concept of moment-matching in frequency domain. It coincides with the coupling matrix-based mode selection criterion used in the OMR method. However, mathematical derivation of moment-matching based mode selection criterion is much simpler than the DtN mapping based derivation used in OMR. Moreover, it does not need the assumption of the special form of external force $Bu(t)$ as used in OMR.

2 Substructuring Methods

In this section, we first discuss a generic CMS method, which is based on the original CMS developed by Hurty [7,8] and Craig and Bampton [4]. Then we specify the difference between the standard CMS method and the new one we propose. We give a justification for the new method in the next section.

In a generic and compact form, the key step of the CMS method is on the construction of the transformation matrix V_n of the form

$$V_n = \begin{array}{c} \\ N_1 \\ N_2 \\ N_3 \end{array} \begin{array}{c} \begin{array}{ccc} n_1 & n_2 & N_3 \end{array} \\ \begin{pmatrix} \Phi_1 & & \Psi_{13} \\ & \Phi_2 & \Psi_{23} \\ & & I_{N_3} \end{pmatrix} \end{array}, \tag{2.7}$$

where $\Psi_{i3} = -K_{ii}^{-1}K_{i3}$ for $i = 1, 2$, and Φ_i is an $N_i \times n_i$ matrix whose columns are the selected n_i eigenvectors $\phi_j^{(i)}$ of the matrix pair (M_{ii}, K_{ii}):

$$K_{ii}\phi_j^{(i)} = \lambda_j^{(i)} M_{ii}\,\phi_j^{(i)} \quad \text{and} \quad (\phi_j^{(i)})^T M_{ii}\phi_k^{(i)} = \delta_{jk}, \tag{2.8}$$

where δ_{jk} is the Kronecker delta. In structural dynamics, Φ_i is the interior partition of the fixed-interface modal matrix and Ψ_{i3} is the interior partition of the constraint-mode matrix.

An orthogonal projection technique for model-order reduction seeks an approximation of $x(t)$ constrained to stay in the subspace spanned by the columns of V_n, namely

$$x(t) \approx V_n z(t)\ .$$

Then by imposing the so-called Galerkin orthogonal condition:

$$MV_n\ddot{z}(t) + KQ_n z(t) - Bu(t) \ \perp\ \text{span}\{V_n\}.$$

it yields a reduced-order system:

$$\Sigma_n : \begin{cases} M_n\ddot{z}(t) + K_n z(t) = B_n u(t) \\ \hat{y}(t) = L_n^T z(t) \end{cases}, \tag{2.9}$$

where $M_n = V_n^T M V_n$, $K_n = V_n^T K V_n$, $B_n = V_n^T B$ and $L_n = V_n^T L$. By the definition of V_n, the matrices M_n and K_n of the reduced system Σ_n are of the following forms

$$M_n = \begin{bmatrix} I & & M_{13}^{(n)} \\ & I & M_{23}^{(n)} \\ (M_{13}^{(n)})^T & (M_{13}^{(n)})^T & \widehat{M}_{33} \end{bmatrix} \text{ and } K_n = \begin{bmatrix} \Lambda_1^{(n)} & & \\ & \Lambda_2^{(n)} & \\ & & \widehat{K}_{33} \end{bmatrix},$$

where

$$M_{i3}^{(n)} = \Phi_i^T \widehat{M}_{i3} \quad \text{and} \quad \widehat{M}_{i3} = M_{i3} - M_{ii}K_{ii}^{-1}K_{i3} \quad \text{for } i = 1, 2,$$

$$\widehat{M}_{33} = M_{33} - \sum_{i=1}^{2}\left(K_{i3}^T K_{ii}^{-1} M_{i3} + M_{i3}^T K_{ii}^{-1} K_{i3} - K_{i3}^T K_{ii}^{-1} M_{ii} K_{ii}^{-1} K_{i3}\right),$$

and $\widehat{K}_{33}$ is the Schur complement of $\text{diag}(K_{11}, K_{22})$ in K of the form

$$\widehat{K}_{33} = K_{33} - K_{13}^T K_{11}^{-1} K_{13} - K_{23}^T K_{22}^{-1} K_{23},$$

and $\Lambda_i^{(n)} = \mathrm{diag}(\lambda_1^{(i)}, \lambda_2^{(i)}, \ldots, \lambda_{n_i}^{(i)})$.

A high-level description of a generic CMS method is as followings.

Generic CMS Method

1. Compute the *selected* eigenpairs $(\lambda_j^{(i)}, \phi_j^{(i)})$ of the generalized eigenproblems $K_{ii}\phi_j^{(i)} = \lambda_j^{(i)} M_{ii}\phi_j^{(i)}$ for $i = 1, 2$,
2. Retain some eigenpairs $(\lambda_j^{(i)}, \phi_j^{(i)})$ to define transformation matrix V_n,
3. Form M_n, K_n, B_n, L_n to define the reduced system Σ_n as in (2.9).

In the standard CMS method, the n_i modes $\phi_j^{(i)}$ associated with smallest eigenvalues $\lambda_j^{(i)}$ are retained to define the projection matrix V_n. V_n is called the *Craig-Bampton transformation matrix* in structure dynamics [3].

In an alternative method, which we call the CMS$_\chi$, the n_i modes $\phi_j^{(i)}$ in V_n are selected according to the highest norm of the rank-one *coupling matrices* $S_j^{(i)}$:

$$S_j^{(i)} = \frac{1}{\lambda_j^{(i)}} \widehat{M}_{i3}^T \phi_j^{(i)} (\phi_j^{(i)})^T \widehat{M}_{i3}. \qquad (2.10)$$

Therefore, the selected modes $\phi_j^{(i)}$ in CMS$_\chi$ may not be in the natural order as in CMS. As a result, to find such n_i modes, we may have to find more than n_i smallest eigenpairs of the matrix pairs (M_{ii}, K_{ii}). This will be shown by numerical examples in section 4. But first we give a justification for the CMS$_\chi$ method in the next section.

3 Derivation of CMS$_\chi$

Let us assume that Φ_i contains all N_i modes of the submatrix pairs (M_{ii}, K_{ii}) for $i = 1, 2$. Then the system Σ_N in its modal coordinate in frequency domain is of the form

$$\left(-\omega^2 \begin{bmatrix} I & & M_{13}^{(N)} \\ & I & M_{23}^{(N)} \\ (M_{13}^{(N)})^T & (M_{23}^{(N)})^T & \widehat{M}_{33} \end{bmatrix} + \begin{bmatrix} \Lambda_1^{(N)} & & \\ & \Lambda_2^{(N)} & \\ & & \widehat{K}_{33} \end{bmatrix} \right) \begin{bmatrix} X_1(\omega) \\ X_2(\omega) \\ X_3(\omega) \end{bmatrix} = \begin{bmatrix} B_1^{(N)} \\ B_2^{(N)} \\ \widehat{B}_3 \end{bmatrix} U(\omega). \qquad (3.11)$$

For the sake of notation, we will drop the superscript $\cdot^{(N)}$ in the rest of section. By solving $X_1(\omega)$ and $X_2(\omega)$ from the first and second equations of (3.11) and then substituting into the third interface equation of the (3.11), it yields

$$\left(\omega^4 \sum_{i=1}^{2} \left[-M_{i3}^T(-\omega I + \Lambda_i)^{-1} M_{i3} \right] - \omega^2 \widehat{M}_{33} + \widehat{K}_{33} \right) X_3(\omega)$$
$$= \left(\omega^2 \sum_{i=1}^{2} \left[M_{i3}^T(-\omega I + \Lambda_i)^{-1} B_i \right] + \widehat{B}_3 \right) U(\omega). \qquad (3.12)$$

In the context of structural dynamics, the equation (3.12) presents the force applied to the interface and applied to it by the subsystems.

Instead of solving equation (3.12) for $X_3(\omega)$ directly, we simplify the equation first, since we are only interested in looking for "important modes". An approximation of (3.12) is taking the first three terms of the power expansion in ω^2 of the coefficient matrix on the left hand side, and taking the constant term on the right hand side. This yields an approximate equation of (3.12):

$$\left[-\omega^4\left(M_{13}^T\Lambda_1^{-1}M_{13} + M_{23}^T\Lambda_2^{-1}M_{23}\right) - \omega^2\widehat{M}_{33} + \widehat{K}_{33}\right]\widetilde{X}_3(\omega) = \widehat{B}_3U(\omega), \quad (3.13)$$

Let the power series expansion of $\widetilde{X}_3(\omega)$ be formally denoted by

$$\widetilde{X}_3(\omega) = \left(\sum_{\ell=0}^{\infty} r_\ell\omega^{2\ell}\right)U(\omega),$$

where r_ℓ are called the ℓ-th moment (vector) of $\widetilde{X}_3(\omega)$. Then by comparing the two sides of equation (3.13) in the power of ω^2, the moments r_ℓ are given by

$$\begin{aligned} r_0 &= \widehat{K}_{33}^{-1}\widehat{B}_3, \\ r_1 &= \widehat{K}_{33}^{-1}\widehat{M}_{33}r_0, \\ r_\ell &= \widehat{K}_{33}^{-1}(\widehat{M}_{33}r_{\ell-1} + (\sum_{i=1}^{2} M_{i3}^T\Lambda_i^{-1}M_{i3})r_{\ell-2}) \quad \text{for } \ell \geq 2. \end{aligned}$$

By an exactly analogous calculation, for the reduced-order system Σ_n in its modal coordinates form, namely

$$M_n = \begin{bmatrix} I & & M_{13}^{(n)} \\ & I & M_{23}^{(n)} \\ (M_{13}^{(n)})^T & (M_{13}^{(n)})^T & M_{33}^{(n)} \end{bmatrix}, \; K_n = \begin{bmatrix} \Lambda_1^{(n)} & & \\ & \Lambda_2^{(n)} & \\ & & K_{33}^{(n)} \end{bmatrix}$$

and

$$B_n = \begin{bmatrix} B_1^{(n)} \\ B_2^{(n)} \\ B_{33}^{(n)} \end{bmatrix}, \; L_n = \begin{bmatrix} L_1^{(n)} \\ L_2^{(n)} \\ L_{33}^{(n)} \end{bmatrix}.$$

The moment vectors $r_\ell^{(n)}$ for the solution $\widetilde{X}_3^{(n)}(\omega)$ of the approximate interface equation are given by

$$\begin{aligned} r_0^{(n)} &= (K_{33}^{(n)})^{-1}B_3^{(n)}, \\ r_1^{(n)} &= (K_{33}^{(n)})^{-1}M_{33}^{(n)}r_0^{(n)}, \\ r_\ell^{(n)} &= (K_{33}^{(n)})^{-1}(M_{33}^{(n)}r_{\ell-1}^{(n)} + (\sum_{i=1}^{2}(M_{i3}^{(n)})^T(\Lambda_i^{(n)})^{-1}M_{i3}^{(n)})r_{\ell-2}^{(n)}) \quad \text{for } \ell \geq 2. \end{aligned}$$

Note that the dimensions of the moment vectors $\{r_\ell\}$ and $\{r_\ell^{(n)}\}$ are the same since we assume that the DOFs of the interface block is unchanged.

A natural optimal strategy to define a reduced-order system Σ_n is to match or approximate as many moments as possible. To match the first moment $r_0 = r_0^{(n)}$, it suggests that

$$K_{33}^{(n)} = \widehat{K}_{33} \quad \text{and} \quad B_3^{(n)} = \widehat{B}_3.$$

To match the second moment $r_1 = r_1^{(n)}$, it derives that

$$M_{33}^{(n)} = \widehat{M}_{33}.$$

Unfortunately, there is no easy way to match the third moment r_2 exactly. Instead, we try to minimize the difference between r_2 and $r_2^{(n)}$:

$$\begin{aligned} \|r_2 - r_2^{(n)}\|_2 &= \|\widehat{K}_{33}^{-1}\left(\sum_{i=1}^{2} M_{i3}^T \Lambda_i^{-1} M_{i3} - (M_{i3}^{(n)})^T (\Lambda_i^{(n)})^{-1} M_{i3}^{(n)}\right)\widehat{K}_{33}^{-1}\widehat{B}_3\|_2 \\ &\le c\|\underbrace{\sum_{j=1}^{N_1} S_j^{(1)} - \sum_{j=1}^{n_1}(S_j^{(1)})^{(n)}}_{1} + \underbrace{\sum_{j=1}^{N_2} S_j^{(2)} - \sum_{j=1}^{n_2}(S_j^{(2)})^{(n)}}_{2}\|_2, \end{aligned} \tag{3.14}$$

where $c = \|\widehat{K}_{33}^{-1}\|_2 \|\widehat{K}_{33}^{-1}\widehat{B}_3\|_2$, a constant independent of the modes $\phi_j^{(i)}$. $S_j^{(i)}$ and $(S_j^{(i)})^{(n)}$ are the coupling matrices for the j-th mode of the subsystem i as defined in (2.10). Assume that $S_j^{(i)}$ and $(S_j^{(i)})^{(n)}$ are in descending order according to their norms, respectively,

$$\|S_1^{(i)}\| \ge \|S_2^{(i)}\| \ge \cdots \ge \|S_{N_i}^{(i)}\|, \quad \|(S_1^{(i)})^{(n)}\| \ge \|(S_2^{(i)})^{(n)}\| \ge \cdots \ge \|(S_{n_i}^{(i)})^{(n)}\|.$$

The best we can do is to set

$$(S_j^{(i)})^{(n)} = S_j^{(i)} \quad \text{for} \quad j = 1, 2, \ldots, n_i.$$

This cancels out the first n_i terms of the differences labled as 1 and 2 of the upper bound in (3.14), and leaves the sums of the remaining terms smallest possible. This yields the CMS$_\chi$-mode selection rule as we described in section 2: *retain the first n_i modes of the subsystem i according to the largest norms of the coupling matrices $S_j^{(i)}$.*

Note that the matrices $\widehat{M}_{i3}$ which couples subsystems and the interface are included in the coupling matrices $S_j^{(i)}$. Therefore, they are reflected for the retention of modes of importance. These coupling effects are essentially ignored by the CMS mode selection. To this end, we also note that CMS$_\chi$-mode selection criterion is essentially the same as the one in the OMR method derived by the DtN mapping [1,6], but without the assumption of the special form of the external force term $Bu(t)$ in the original system Σ_N (1.1).

4 Numerical Experiments

In this section, we present two numerical examples to compare the two mode selection criteria discussed in this paper. All numerical experiments were run in MATLAB on a Linux Server with Dual 1.2Ghz CPUs and 2GB of memory.

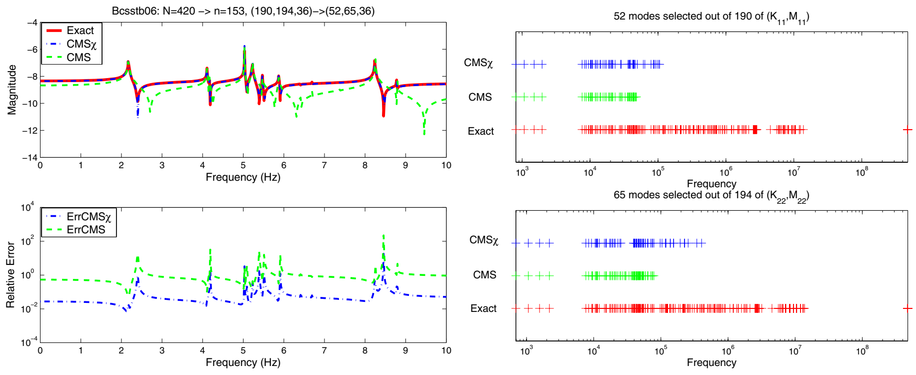


Fig. 2. Left: magnitudes (in log of base 10) of the transfer functions (top) and relative errors (bottom). Right: retained modes of subsystems by CMS and CMS_χ

Example 1. In this example, the mass and stiffness matrices M and K are from Harwell-Boeing BCS sparse matrix collection [5]. The number of DOFs of Σ_N is $N = 420$, and that of two subsystems are $N_1 = 190$ and $N_2 = 194$, respectively. The top left plot of Fig. 2 shows the magnitude (in log of base 10) of the transfer function $H(\omega)$ of the SISO system Σ_N with $B = L = [\ 1\ 0\ \dots\ 0\ ]^T$. The transfer functions $H_n^{\mathrm{CMS}}(\omega)$ and $H_n^{\mathrm{CMS}_\chi}(\omega)$ of the reduced systems Σ_n, computed by CMS and CMS_χ, are shown in the same plot. The number of DOFs of reduced-order systems Σ_n is $n = 153$ with $n_1 = 52$ and $n_2 = 65$, respectively. The relative errors $|H(\omega) - H_n^{\mathrm{CMS}}(\omega)|/|H(\omega)|$ and $|H(\omega) - H_n^{\mathrm{CMS}_\chi}(\omega)|/|H(\omega)|$ shown in the lower left plot of Fig. 2 indicate that $H_n^{\mathrm{CMS}_\chi}(\omega)$ is a much accurate approximation of $H(\omega)$ than $H_n^{\mathrm{CMS}}(\omega)$, under the same order of reduced DOFs.

Two right plots of Fig. 2 show the eigenvalues of original systems and the ones retained by CMS and CMS_χ. Note again that the numbers of eigenvalues of subsystems retained by the two methods are the same. CMS_χ skips some of lower frequency eigenvalues, and uses some higher frequency eigenvalues to take into the account of coupling effects between the subsystems and the interface. On the other, CMS simply takes the lowest frequency eigenvalues in order.

Example 2. This is a SISO system Σ_N arised from simulation of a prototype radio-frequency MEMS filter [2]. The DOFs of Σ_N is $N = 490$ and that of two subsystems are $N_1 = N_2 = 238$. The DOFs of interface is $N_3 = 14$. Fig. 3 shows the transfer functions $H(\omega)$, $H_n^{\mathrm{CMS}}(\omega)$ and $H_n^{\mathrm{CMS}_\chi}(\omega)$. The DOFs of reduced subsystems by the both methods are $n_1 = n_2 = 85$. The relative errors $|H(\omega) - H_n^{\mathrm{CMS}}(\omega)|/|H(\omega)|$ and $|H(\omega) - H_n^{\mathrm{CMS}_\chi}(\omega)|/|H(\omega)|$ in the lower left plot of Fig. 3 show the improvement made by the new CMS_χ method. Two right plots of Fig. 3 show the differences in the retention of the same number of modes of subsystems.

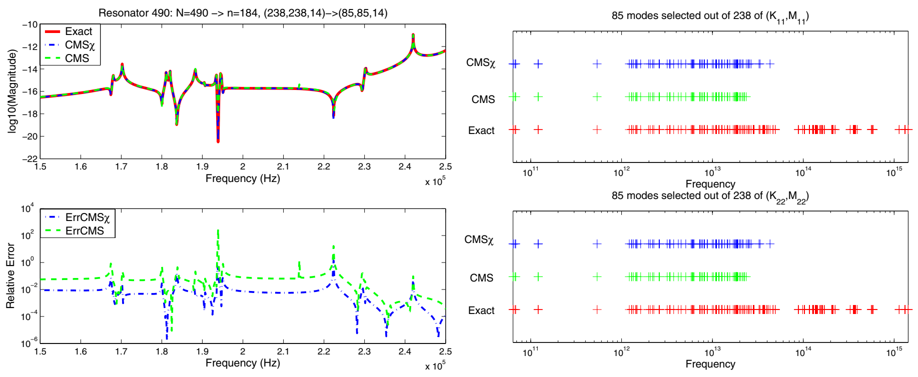


Fig. 3. Left: magnitudes (in log of base 10) of the transfer functions (top) and relative errors (bottom). Right: retained modes of subsystems by CMS and CMS_χ

5 Conclusion Remarks

A new coupling matrix-based mode selection criterion for the popular CMS method is presented in this paper. It is derived based on moment-matching property for the interface solution. Our work is motivated by the recent work of Givoli *et al* [1,6], in which the term "coupling matrix" is coined. Our mode selection criterion is compatible to the one proposed by Givoli *et al*, which uses Dirichlet-to-Neumann (DtN) mapping as an analysis tool. The performance improvement of the new mode selection criterion is demonstrated by numerical examples.

The coupling matrix-based mode selection costs more than the standard one, since some extra eigenpairs of the subsystems are typically required. If the sizes of subsystems are moderate, the extra cost may not be significant measured by the CPU time. Multilevel substructuring with an optimal mode selection is a subject of future study. It is worth to note that modal reduction methods as discussed in this paper are generally less accurate and efficient than Krylov subspace-based reduction methods. A Krylov subspace-based substructuring method is in progress.

Acknowledgments

The work is supported in part by the National Science Foundation under grant No. 0220104.

References

1. P.E. Barbone and D. Givoli. Optimal modal reduction of vibrating substructures. *Int. J. Numer. Meth. Engng*, 57:341–369, 2003.
2. D. Bindel, Z. Bai, and J. Demmel. Reduced order models in Microsystems and RF MEMS. To appear in the proceedings of PARA'04: Workshop on the state-of-the-art in scientific computing, Lyngby, Denmark, June 20-23, 2004.
3. R. R. Craig Jr. Coupling of substructures for dynamic analysis - an overview. *AIAA-2000-1573*, 2000.
4. R. R. Craig Jr. and M. C. C. Bampton. Coupling of substructures for dynamic analysis. *AIAA Journal*, 6(7):1313–1319, 1968.
5. I.S. Duff, R.G. Grimes, and J.G. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection (release 1). Technical Report RAL-92-086, Rutherford Appleton Laboratory, December 1992. Available at the MatrixMarket: http://math.nist.gov/MatrixMarket.
6. D. Givoli, P. E. Barbone, and I. Patlashenko. Which are the important modes of a subsystem? *Int. J. Numer. Meth. Engng*, 59:1657–1678, 2004.
7. W. C. Hurty. Vibrations of structural systems by component-mode synthesis. *Journal of the Engineering Mechanics Division, ASCE*, 86:59–69, 1960.
8. W. C. Hurty. Dynamic analysis of structural systems using component modes. *AIAA Journal*, 3:678–685, 1965.
9. J.B. Keller and D. Givoli. Exact non-reflecting boundary conditions. *Journal of Computational Physics*, 82:172–192, 1989.

Model Reduction for RF MEMS Simulation

David S. Bindel[1], Zhaojun Bai[2], and James W. Demmel[3]

[1] Department of Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, CA 94704
dbindel@eecs.berkeley.edu

[2] Department of Computer Science
University of California at Davis
Davis, CA 95616
bai@cs.ucdavis.edu

[3] Department of Electrical Engineering and Computer Science and
Department of Mathematics
University of California at Berkeley
Berkeley, CA 94704
demmel@eecs.berkeley.edu

Abstract. Radio-frequency (RF) MEMS resonators, integrated into CMOS chips, are of great interest to engineers planning the next generation of communication systems. Fast simulations are necessary in order to gain insights into the behavior of these devices. In this paper, we discuss two structure-preserving model-reduction techniques and apply them to the frequency-domain analysis of two proposed MEMS resonator designs.

1 Introduction

Radio-frequency (RF) electromechanical resonators used in frequency references and filters are critical to modern communication systems. For RF engineers using quartz, ceramic, and surface acoustic wave devices, surface-micromachined MEMS resonators in development offer an attractive alternative. Because they can be integrated into CMOS, MEMS resonators have the potential to cost less area, power, and money than existing alternatives [10]. Such integrated high-frequency filters could qualitatively change the design of cell phones, making it possible to build a cheap phone to work with multiple networks operating at different carrier frequencies, and lowering power and size constraints to the point that inexpensive "Dick Tracy" watch phones could be a reality.

Researchers working on integrated MEMS resonators currently rely on a trial-and-error design process. There is a clear need for quick, accurate simulations to minimize the labor and fabrication costs of trial-and-error design. Lumped-parameter models and simple modal analyses do not adequately describe subtle continuum-level effects and mode interactions which strongly affect device behavior. We therefore rely on model-reduction methods to perform frequency-response computations quickly.

Energy losses in high-frequency resonators are critical to designers. The *quality factor* (Q), a ratio of energy stored in a resonator to energy lost in each cycle, must be high for a resonator to be useful in a filter or frequency reference. Despite their importance, the details of energy loss in MEMS resonators are not well understood, and accurate loss modeling remains an area of active research. Among other mechanisms, energy can be lost through viscous damping, through thermoelastic effects, or through elastic waves carrying energy away into the substrate [8]. Each mechanism leads to a different structure in the model equations. A mechanical system subject to viscous damping is modeled by a second-order system of equations with a nonzero first-order term; the thermoelastic equations are coupled first- and second-order equations; while infinite-domain problems can be modeled by a coordinate transformation resulting in complex symmetric mass and stiffness matrices. Here we consider viscous and anchor losses. Viscous loss models are relatively common, but to our knowledge, ours is the first physical model of anchor loss in a bulk-mode MEMS resonator.

In this paper, we describe two model reduction techniques for computing the frequency response of MEMS resonators. We first describe how to reduce second order systems using a *Second-Order Arnoldi* (SOAR) algorithm which preserves the second-order structure of the equations; we use this technique to simulate a checkerboard-shaped mechanical filter. We then describe how we model infinite-domain problems using a *perfectly-matched layer* (PML), and describe a model-reduction technique which preserves the structure of the PML; we use this technique to study the frequency response of a disk resonator.

2 Model Reduction for Second-Order Systems

2.1 Second-Order Systems and SOAR

Since they are partly mechanical, most MEMS models are naturally second-order in time. Depending on the device, there may also be equations which are first order in time or algebraic, such as the equations for heat transport or for electrostatics. The linearized equations for the coupled system can be written as a continuous time-invariant single-input single-output second-order system

$$\Sigma_N : \begin{cases} M\ddot{q}(t) + D\dot{q}(t) + Kq(t) = b\,u(t) \\ \qquad\qquad\qquad\quad y(t) = l^T q(t) \end{cases} \tag{2.1}$$

with initial conditions $q(0) = q_0$ and $\dot{q}(0) = \dot{q}_0$. Here t is time; $q(t) \in \mathcal{R}^N$ is a vector of state variables; N is the state-space dimension; $u(t) \in \mathcal{R}$ and $y(t) \in \mathcal{R}$ are the input force and output measurement functions; M, D, $K \in \mathcal{R}^{N\times N}$ are system mass, damping, and stiffness matrices; and b and l are input distribution and output measurement vectors. The state-space dimension N of the system Σ_N is typically large and it can be slow to use for practical analysis and design. Therefore, model-order reduction techniques are needed to construct compact models which both retain important properties of the original model and are fast to simulate.

A common approach to model-order reduction for the second-order model Σ_N is to add variables for the velocities $\dot{q}(t)$, and so create a first-order model of twice the dimension of the original. Then the first-order model is reduced. However, the reduced first-order model may not correspond to any second-order model, and may lack properties of the original model, such as stability and passivity. There have been a number of efforts toward such structure-preserving model-order reduction methods; we focus on a subspace projection method based on a second-order Arnoldi (SOAR) procedure that not only preserves the second-order structure, but also matches the same number of moments as the standard Arnoldi-based Krylov subspace method via linearization for about the same amount of work. For the rest of this subsection, we present a SOAR-based model-order reduction method from the view discussed in [2].

The second-order system Σ_N is represented in the time domain in (2.1). Equivalently, one can represent the system in the frequency domain via the Laplace transform. Assuming homogeneous initial conditions, the frequency-domain input $U(s)$ and output $Y(s)$ are related by the *transfer function*

$$H(s) = l^T(s^2M + sD + K)^{-1}b = \sum_{\ell=0}^{\infty} m_\ell (s - s_0)^\ell, \tag{2.2}$$

where the coefficients m_ℓ in the Taylor series about s_0 are called *moments*. The moments can be written as $m_\ell = l^T r_\ell$, where the *second-order Krylov vector sequence* $\{r_\ell\}$ is defined by the following recurrence:

$$\begin{aligned} r_0 &= \widehat{K}^{-1}b \\ r_1 &= -\widehat{K}^{-1}\widehat{D}r_0 \\ r_\ell &= -\widehat{K}^{-1}(\widehat{D}r_{\ell-1} + Mr_{\ell-2}) \quad \text{for} \quad \ell = 2, 3, \ldots \end{aligned} \tag{2.3}$$

where $\widehat{K} = s_0^2M + s_0D + K$ and $\widehat{D} = 2s_0M + D$.

The subspace $\mathcal{G}_n = \text{span}\{r_\ell\}_{\ell=0}^{n-1}$ is called a *second-order Krylov subspace*. Let Q_n be an orthonormal basis of $\mathcal{G}_n$. We seek an approximation $q(t) \approx Q_n z(t) \in \mathcal{G}_n$; this is often referred to as a *change of state coordinates*. By imposing the Galerkin condition:

$$MQ_n\ddot{z}(t) + DQ_n\dot{z}(t) + KQ_nz(t) - b\,u(t) \perp \mathcal{G}_n,$$

we obtain the following structure-preserving reduced-order model:

$$\Sigma_n : \begin{cases} M_n\ddot{z}_n(t) + D_n\dot{z}_n(t) + K_nz(t) = b_n\,u(t) \\ \tilde{y}(t) = l_n^T z(t) \end{cases}, \tag{2.4}$$

where $M_n = Q_n^TMQ_n$, $D_n = Q_n^TDQ_n$, $K_n = Q_n^TKQ_n$, $b_n = Q_n^Tb$ and $l_n = Q_n^Tl$. It can be shown that the first n moments of the reduced model Σ_n agree with those of the original model Σ_N. Furthermore, if M, D, and K are symmetric and $b = l$, then the first $2n$ moments of the models are the same. This method has the same order of approximation as the standard Arnoldi-based Krylov subspace method via linearization.

We produce an orthonormal basis Q_n for the second-order Krylov subspace $\mathcal{G}_n$ using a Second-Order ARnoldi (SOAR) procedure proposed by Su and Craig [12] and further improved by Bai and Su [3]. At step j, the algorithm computes

$$r = -\widehat{K}^{-1}(\widehat{D}q_j + Mw) \tag{2.5}$$

where w is an auxiliary vector computed at the end of the previous step. Then r is orthogonalized against Q_j and normalized to produce q_{j+1} and w is recomputed.

In contrast to the standard Arnoldi algorithm, the transformed matrix triplet $(M, \widehat{D}, \widehat{K})$ is used to generate an orthonormal basis Q_n of $\mathcal{G}_n$, but the original matrix triplet (M, D, K) is directly projected onto the subspace $\mathcal{G}_n$ to define the reduced-order model Σ_n. By explicitly formulating the matrices M_n, D_n and K_n, essential structures of M, D and K are preserved. For example, if M is symmetric positive definite, so is M_n. As a result, we can preserve the stability, symmetry and physical meaning of the original second-order model Σ_N. This is in the same spirit as the widely-used PRIMA algorithm for passive reduced-order modeling of linear dynamical systems arising from interconnect analysis in circuit simulations [11].

The SOAR-based model-order reduction algorithm has many desirable properties compared to the linearization approach. The reduced system Σ_n not only preserves the second-order structure, but also matches the same number of moments as the standard method of projecting a linearized system onto a basis of n Arnoldi vectors. SOAR-based algorithms require less space and fewer flops (for a subspace of the same dimension), and also provide more accurate results.

2.2 Modeling of a Checkerboard Resonator

As an application, we build a reduced-order model from a finite element simulation of a prototype MEMS filter. The goal for this device is to produce a high-frequency bandpass filter to replace, for example, the surface acoustic wave (SAW) devices used in cell phones. The device (Figure 1) consists of a checkerboard of silicon squares which are linked at the corners [6]. The "checkers" are held at a fixed voltage bias relative to the drive and sense electrodes placed around the perimeter. A radio-frequency (RF) voltage variation on drive electrodes at the northwest corner creates an electrostatic force which causes the device to move in plane. The motion induces changes in capacitance at the sense electrodes at the southwest corner of the device. The induced motion is typically very small; the checker squares are two microns thick and tens of microns on a side, and the maximum displacement is on the order of tens of nanometers.

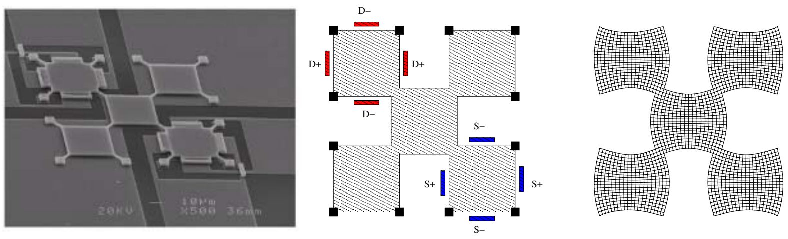


Fig. 1. Illustration of a checkerboard resonator. The SEM picture (left) shows a fabricated device, and the simulation (right) shows one resonant mode excited during operation. The motion is excited at the northwest corner and sensed at the southeast corner (center)

A single square pinned at the corners exhibits a "Lamé mode." If the interaction between squares was negligible, the system would have a five-fold eigenvalue corresponding to independent Lamé-mode motions for each square. The coupling between

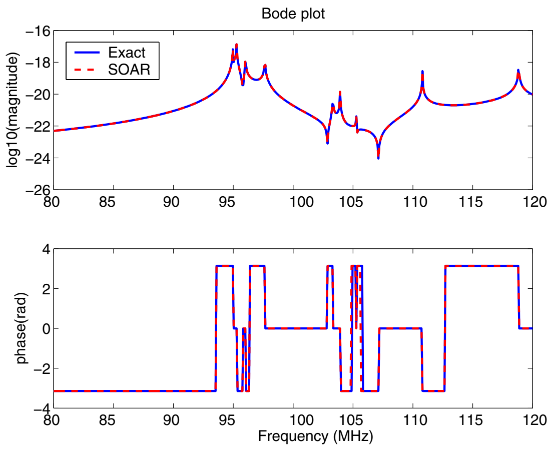


Fig. 2. Bode plots from finite element model and reduced order model for a 3-by-3 checkerboard resonator

the squares causes the five eigenvalues to split, so there are several poles near each other; consequently, the array has low motional resistance near the target frequency. The same idea of using weakly coupled arrays has also been applied to other RF resonator designs [9].

We model the checkerboard with linear 2D plane stress elements and an empirical viscous damping model. Even for a small mesh ($N = 3231$), model reduction is beneficial. Using a reduced model with 150 degrees of freedom, we obtain a Bode plot which is visually indistinguishable from the plot for the unreduced system (Figure 2). We have also created a visualization tool which designers can use to see the forced motion at different frequencies (Figure 3). With a reduced model, we can compute the shape of the motion at a specified frequency within tens of milliseconds instead of seconds, quickly enough for the user to interactively scan through frequencies of interest. Designers can use these visualizations to build intuition about different strategies for constructing anchors, connecting resonator components, and placing drive and sense electrodes.

3 Model Reduction for Perfectly-Matched Layers

3.1 Perfectly-Matched Layers and Symmetry-Preserving Projection

In several high MHz or GHz frequency resonator designs, the dominant loss mechanism appears to be radiation of elastic energy through anchors. In these designs, the resonating device is so much smaller than the silicon substrate that waves leaving the anchor are so attenuated by material losses as to render negligible any reflections from the sides of the chip. That is, the bulk of the chip can be modeled as a semi-infinite medium. We model this semi-infinite domain using a *perfectly matched layer* (PML), which absorbs waves at any angle of incidence [4].

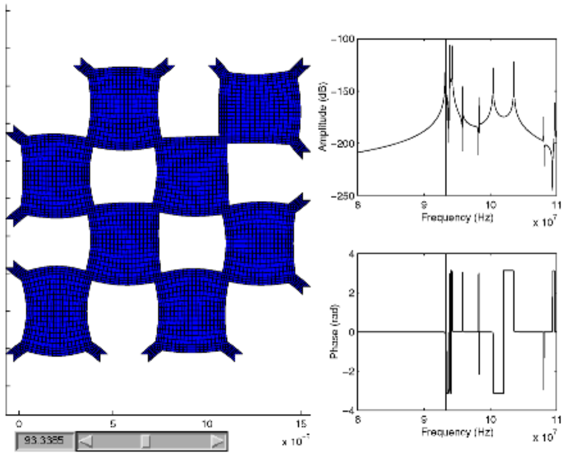


Fig. 3. Screenshot of a visualization tool for observing forced response shapes for in-plane resonators. Using a reduced model, the tool can compute and plot the response shape quickly enough for interactive use

Bérenger invented the perfectly matched layer for problems in electromagnetics [5], but it was almost immediately re-interpreted as a complex-valued change of coordinates which can be applied to any linear wave equation [14,13]. The weak form of the PML equations for time-harmonic linear elasticity [4] is

$$\int_{\Omega} \tilde{\epsilon}(w)^T D \tilde{\epsilon}(u) \tilde{J}\, d\Omega - \omega^2 \int_{\Omega} \rho w \cdot u \tilde{J}\, d\Omega = \int_{\Gamma} w \cdot t\, d\Gamma \tag{3.6}$$

where u and w are displacement and weight functions on the domain Ω; t is a traction defined on the boundary Γ; ρ and D are the mass density and the elasticity matrix; $\tilde{\epsilon}$ is a transformed strain vector; and $\tilde{J}$ is the Jacobian determinant of the PML transformation. This weak form is nearly identical to the standard weak form of linear elasticity, and when we discretize with finite elements, we have the familiar system

$$(K - \omega^2 M)q = b \tag{3.7}$$

where K and M are now *complex* symmetric.

To make the attenuation through the PML frequency-independent, the coordinate transformations in [4] are frequency-dependent. Using a frequency-dependent transformation, the transfer function is given by

$$H(i\omega) = l^T \left(K(\omega) - \omega^2 M(\omega)\right)^{-1} b \tag{3.8}$$

and complex resonances satisfy the nonlinear eigenvalue equation

$$\det\left(K(\omega) - \omega^2 M(\omega)\right) = 0. \tag{3.9}$$

However, when the frequency range of interest is not too wide – when the maximum and minimum frequency considered are within a factor of two or three of each other –

the parameters of the PML may be chosen once to give acceptable attenuation over the desired frequency range. So for ω near enough to ω_0, we approximate $H(i\omega)$ by

$$H_0(i\omega) = l^T \left(K(\omega_0) - \omega^2 M(\omega_0)\right)^{-1} b. \tag{3.10}$$

Similarly, the nonlinear eigenvalue problem (3.9) may be replaced by a (generalized) linear eigenvalue problem.

In [1], Arbenz and Hochstenbach suggest a variant of the Jacobi-Davidson algorithm for complex symmetric eigenvalue problems. In this method, the standard Rayleigh quotient estimate of the eigenvalue, $\theta(v) = (v^H K v)/(v^H M v)$, is replaced by $\psi(v) = (v^T K v)/(v^T M v)$. Assuming $v^T M v \neq 0$, $\psi(v)$ converges quadratically to an eigenvalue of a complex-symmetric pencil as v approaches an eigenvector, while $\theta(v)$ converges only linearly. The extra order of accuracy comes from the symmetry of the quadratic forms

$$u^T M v = v^T M u \text{ and } u^T K v = v^T K u.$$

We wish to maintain this symmetry in our reduced-order models as well.

To build a reduced-order model, we generate a basis V for the Krylov subspace $\mathcal{K}_n((K(\omega_0) - \omega_0^2 M(\omega_0))^{-1}, b)$ with the standard Arnoldi algorithm. We then construct a symmetry-preserving reduced-order model by choosing an orthonormal projection basis W such that

$$\operatorname{span}(W) = \operatorname{span}([\operatorname{Re}(V), \operatorname{Im}(V)]). \tag{3.11}$$

The reduced model will be at least as accurate as the usual Arnoldi projection, but will maintain the complex symmetry of the original system. This reduced model also corresponds to a Bubnov-Galerkin discretization of the PML equation with a set of real-valued global shape functions.

3.2 Modeling of a Disk Resonator

As an example of PML model reduction, we study the anchor loss in a disk-shaped MEMS resonator presented in [15] (Figure 4). The disk sits on a silicon post, and differences in voltage between the disk and the surrounding drive electrodes pull the disk rim outward, exciting an axisymmetric motion. The disk is driven near the frequency of

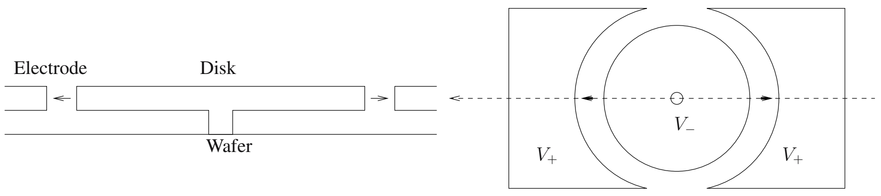


Fig. 4. Schematic of the Michigan disk resonator. An overhead view (right) shows the arrangement of the resonating disk and the electrodes which force it. An idealized cross-section (left) is used in an axisymmetric simulation, where the wafer substrate is treated as semi-infinite using a perfectly matched layer

the first or second radial mode, and the motion is sensed capacitively by electrodes on the surface of the substrate.

We model this resonator using axisymmetric finite elements, with a PML region to mimic the effects of a semi-infinite substrate. We wish to use this model to study sharpness of resonant peaks, as quantified by the quality factor Q. For a resonant frequency $\omega = \alpha + i\beta$, Q is given by $\alpha/(2\beta)$. For typical RF applications, Q values of at least 1000 are required; higher values are better.

Because the discretization errors in the real and imaginary parts of the computed eigenvalues are about the same size, we must resolve ω to an overall relative error significantly less than Q^{-1} before the computed Q converges. Consequently, we use a relatively fine mesh of high-order (bicubic) elements with 57475 unknowns in order to resolve the behavior of this device. With this mesh, the computed Q value of the second radial mode was 6250; the measured value of an actual device was 7330 (in vacuum).

Because we expect a single sharp peak to dominate the response in the frequency range of interest, we might expect a single step of shift-invert Arnoldi with a good shift to produce a reasonable approximation to $H_0(i\omega)$. If the device possesses two nearly coincident modes, then two Arnoldi iterations are needed. When the two modes are very close, they strongly interact, and any reduced model must account for both. Furthermore, when multiple poles are involved, the Q values provided by eigenvalue computations no longer provide a complete picture of the behavior near a resonant peak. To understand the response, we use a reduced model with three Arnoldi vectors (two steps of Arnoldi plus a starting vector), which approximates the full model very closely near the peak (Figure 5). Though the peak for one of the modes has negligible magnitude compared to the peak for the other mode, the interaction of the two modes strongly affects the sharpness of the dominant mode peak: for values of the disk thickness where the two modes most closely coincide, the computed Q value for the dominant-mode peak varies over five orders of magnitude [7].

4 Conclusion

In this paper, we have described two model reduction techniques: a method based on the SOAR (Second-Order ARnoldi) algorithm, which preserves the second-order structure of a full system; and a method which uses a standard Arnoldi basis in a novel way to preserve the complex-symmetric structure of an infinite-domain problem approximated by a perfectly matched layer. We have illustrated the utility of these methods by computing the frequency-response behavior of two real designs of very high-frequency MEMS resonators. In each case, the interactions between multiple resonant modes proves critical, so that naive projection onto a single mode is inadequate for exploring even the local frequency-response behavior.

As we continue to study energy loss mechanisms in RF MEMS, we expect model reduction to play an even more critical role. For example, to study anchor loss in the checkerboard resonator, we plan to build a 3D finite element model with perfectly matched layers below the anchors; this model will be much larger than the 2D model described in this paper. We are also interested in structure-preserving model reduction for thermoelastic damping, which is an important loss mechanism in at least some

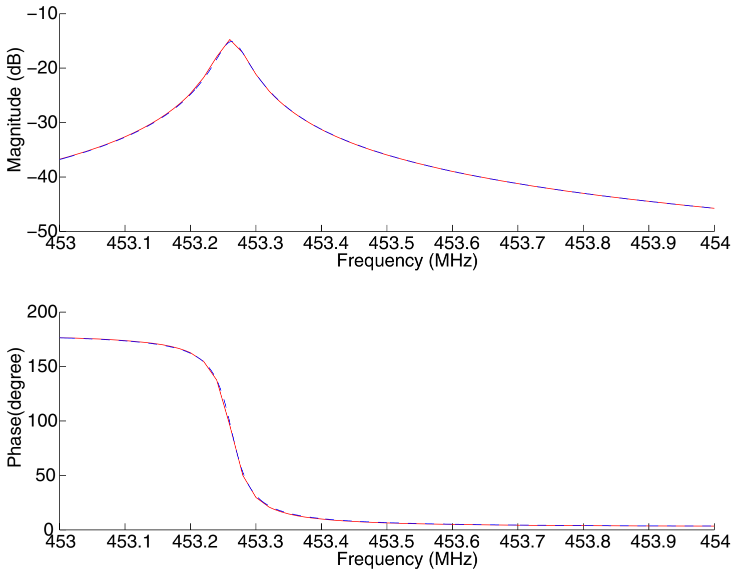


Fig. 5. Bode plot for a disk resonator with nearly coincident modes. The exact model (solid) matches the reduced model (dashed) produced from two Arnoldi steps

types of flexural resonators [8]. Finally, to more quickly study how variations in device geometry affect performance, we plan to investigate methods for building parameterized reduced-order models and reusable reduced-order models for substructures.

Acknowledgements

Sunil Bhave and Roger Howe of the Berkeley EECS department proposed and fabricated the checkerboard resonator design, and have provided valuable feedback during the development of our simulations. The study of resonator loss mechanisms is collaborative work with Sanjay Govindjee and Tsuyoshi Koyama of the Berkeley Civil Engineering department.

References

1. P. Arbenz and M. E. Hochstenbach. A Jacobi-Davidson method for solving complex symmetric eigenvalue problems. *SIAM J. Sci. Comp.*, 25(5):1655–1673, 2004.
2. Z. Bai and Y. Su. Dimension reduction of second-order dynamical systems via a second-order Arnoldi method. *SIAM J. Sci. Comp.*, 2004. to appear.
3. Z. Bai and Y. Su. SOAR: A second-order Arnoldi method for the solution of the quadratic eigenvalue problem. *SIAM J. Matrix Anal. Appl.*, 2004. to appear.
4. U. Basu and A. Chopra. Perfectly matched layers for time-harmonic elastodynamics of unbounded domains: theory and finite-element implementation. *Computer Methods in Applied Mechanics and Engineering*, 192:1337–1375, 2003.

5. J.-P. Bérenger. A perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, 114(2):185–200, 1994.
6. S. Bhave, D. Gao, R. Maboudian, and R. T. Howe. Fully differential poly-SiC Lamé mode resonator and checkerboard filter. In *Proceedings of MEMS 05*, Miami, FL, January 2005.
7. D. S. Bindel, E. Quévy, T. Koyama, S. Govindjee, J. W. Demmel, and R. T. Howe. Anchor loss simulation in resonators. In *Proceedings of MEMS 05*, Miami, FL, January 2005.
8. R. N. Candler, H. Li, M. Lutz, W.-T. Park, A. Partridge, G. Yama, and T. W. Kenny. Investigation of energy loss mechanisms in micromechanical resonators. In *Proceedings of Transducers 03*, pages 332–335, Boston, June 2003.
9. M. U. Demirci, M. A. Abdelmoneum, and C. T.-C. Nguyen. Mechanically corner-coupled square microresonator array for reduced series motional resistance. In *Proc. of the 12th Intern. Conf. on Solid State Sensors, Actuators, and Microsystems*, pages 955–958, Boston, June 2003.
10. C. T.-C. Nguyen. Vibrating RF MEMS for low power wireless communications. In *Proceedings of the 2001 International MEMS Workshop (iMEMS01)*, pages 21–34, Singapore, July 2001.
11. A. Odabasioglu, M. Celik, and L. T. Pileggi. PRIMA: passive reduced-order interconnect macromodeling algorithm. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 17:645–654, 1998.
12. T.-J. Su and R. R. Craig Jr. Model reduction and control of flexible structures using Krylov vectors. *J. of Guidance, Control and Dynamics*, 14:260–267, 1991.
13. F. Teixeira and W. Chew. Complex space approach to perfectly matched layers: a review and some new developments. *International Journal of Numerical Modelling*, 13(5):441–455, 2000.
14. E. Turkel and A. Yefet. Absorbing PML boundary layers for wave-like equations. *Applied Numerical Mathematics*, 27(4):533–557, 1998.
15. J. Wang, Z. Ren, and C. T-C. Nguyen. Self-aligned 1.14 GHz vibrating radial-mode disk resonators. In *Proceedings of Transducers 03*, pages 947–950, 2003.

A Model-Order Reduction Technique for Low Rank Rational Perturbations of Linear Eigenproblems

Frank Blömeling and Heinrich Voss

Department of Mathematics
Hamburg University of Technology
D-21071 Hamburg, Germany
{f.bloemeling,voss}@tu-harburg.de

Abstract. Large and sparse rational eigenproblems where the rational term is of low rank k arise in vibrations of fluid–solid structures and of plates with elastically attached loads. Exploiting model order reduction techniques, namely the Padé approximation via block Lanczos method, problems of this type can be reduced to k–dimensional rational eigenproblems which can be solved efficiently by safeguarded iteration.

1 Introduction

In this paper we consider the rational eigenvalue problem

$$T(\lambda)x := -Kx + \lambda Mx + CD(\lambda)C^T x = 0, \tag{1.1}$$

where $K \in \mathbb{R}^{N\times N}$ and $M \in \mathbb{R}^{N\times N}$ are sparse symmetric and positive (semi-) definite matrices, $C \in \mathbb{R}^{N\times k}$ is a rectangular matrix of low rank k, and $D(\lambda) \in \mathbb{R}^{k\times k}$ is a real diagonal matrix depending rationally on a real parameter λ. Problems of this type arise in (finite element models of) vibrations of fluid–solid structures and of plates with elastically attached loads, e.g.

Problem (1.1) has a countable set of eigenvalues which can be characterized as minmax values of a Rayleigh functional [8], and its eigenpairs can be determined by iterative projection methods of Arnoldi [9] or Jacobi–Davidson type [2].

In this paper we take advantage of the fact that problem (1.1) can be interpreted as a rational perturbation of small rank k of a linear eigenproblem. Decomposing $x \in \mathbb{R}^N$ into its component in the range of C and its orthogonal complement, (1.1) can be rewritten as

$$\tilde{T}(\lambda)\tilde{x} := D(\lambda)^{-1}\tilde{x} + C^T(-K + \lambda M)^{-1}C\tilde{x} = 0, \tag{1.2}$$

which is a rational eigenvalue problem of much smaller dimension k.

The eigenproblem (1.2) retains the symmetry properties of problem (1.1), and hence, in principle it can be solved efficiently by safeguarded iteration. However, every step of safeguarded iteration requires the evaluation of $\tilde{T}(\lambda)$ for some λ, i.e. of $C^T(-K + \lambda M)^{-1}C$, which is too expensive because the dimension N is very large.

The term $C^T(-K + \lambda M)^{-1}C$ appears in transfer functions of time invariant linear systems, and in systems theory techniques have been developed to reduce the order of this term considerably. Taking advantage of these techniques, namely of the Padé approximation via the block Lanczos method, problem (1.2) is replaced by a problem of the same structure of much smaller order. Since this approximating problem inherits the symmetry properties of the original problem it can be solved efficiently by safeguarded iteration.

This paper is organized as follows. Section 2 presents the rational eigenproblems governing free vibrations of a plate with elastically attached loads, and of a fluid–solid structure. Section 3 summarizes the minmax theory for nonoverdamped nonlinear symmetric eigenvalue problems, and recalls the safeguarded iteration for determining eigenpairs of dense problems of this type in a systematic way. Section 4 discusses the reduction of problem (1.1) to a k dimensional rational eigenproblem and the application of the Padé–by–Lanczos method to reduce the order of the rational term. We demonstrate that the eigenvalues of the reduced problem allow a minmax characterization. Hence, it can be solved in a systematic way and efficiently by safeguarded iteration. Section 5 reports on the numerical behaviour of the model–order reduction technique for a finite element discretization of a plate problem with elastically attached loads. It demonstrates that for this type of problems the method is superior to iterative projection methods like Arnoldi's method [9].

2 Rational Eigenvalue Problems

In this section we briefly present two examples of rational eigenproblems of type (1.1).

Consider an isotropic thin plate occupying the plane domain Ω, and assume that for $j = 1, \ldots, p$ at points $\xi_j \in \Omega$ masses m_j are joined to the plate by elastic strings with stiffness coefficients k_j.

Then the the flexurable vibrations are governed by the eigenvalue problem

$$Lu(\xi) = \omega^2 \rho d u + \sum_{j=1}^{p} \frac{\omega^2 \sigma_j}{\sigma_j - \omega^2} m_j \delta(\xi - \xi_j) u \ , \ \xi \in \Omega \tag{2.3}$$

$$Bu(\xi) = 0 \ , \ \xi \in \partial\Omega \tag{2.4}$$

where $\rho = \rho(\xi)$ is the volume mass density, $d = d(\xi)$ is the thickness of the plate at a point $\xi \in \Omega$, and $\sigma_j = \frac{k_j}{m_j}$. B is some boundary operator depending on the support of the plate, δ denotes Dirac's delta distribution, and

$$L = \partial_{11} D(\partial_{11} + \nu \partial_{22}) + \partial_{22} D(\partial_{22} + \nu \partial_{11}) + 2\partial_{12} D(1 - \nu)\partial_{12}$$

is the plate operator where $\partial_{ij} = \partial_i \partial_j$ and $\partial_i = \partial/\partial \xi_i$, $D = Ed^3/12(1 - \nu^2)$ the flexurable rigidity of the plate, E is Young's modulus, and ν the Poisson ratio.

Discretizing by finite elements yields a matrix eigenvalue problem

$$-Kx + \lambda M x + \sum_{j=1}^{p} \frac{\lambda \sigma_j}{\sigma_j - \lambda} e_{i_j} e_{i_j}^T x = 0,$$

which can be easily given the form (1.1). Here $\lambda = \omega^2$ and $u(\xi_j) = x_{i_j}$

Another rational eigenproblem of type (1.1) is governing free vibrations of a tube bundle immersed in a slightly compressible fluid under the following simplifying assumptions: The tubes are assumed to be rigid, assembled in parallel inside the fluid, and elastically mounted in such a way that they can vibrate transversally, but they can not move in the direction perpendicular to their sections. The fluid is assumed to be contained in a cavity which is infinitely long, and each tube is supported by an independent system of springs (which simulates the specific elasticity of each tube). Due to these assumptions, three-dimensional effects are neglected, and so the problem can be studied in any transversal section of the cavity. Considering small vibrations of the fluid (and the tubes) around the state of rest, it can also be assumed that the fluid is irrotational.

Let $\Omega \subset \mathbb{R}^2$ denote the section of the cavity, and $\Omega_j \subset \Omega, j = 1, \dots, p$, the sections of the tubes. Then the free vibrations of the fluid are governed by (cf. [3])

$$
\begin{aligned}
&c^2 \Delta\phi + \omega^2\phi = 0 \quad \text{in } \Omega \setminus \cup_{j=1}^{p} \Omega_j \\
&\frac{\partial\phi}{\partial n} = \frac{\rho_0\omega^2}{k_j - \omega^2 m_j} n \cdot \int\limits_{\partial\Omega_j} \phi n \, ds \quad \text{on } \partial\Omega_j, \ j = 1, \dots, p \\
&\frac{\partial\phi}{\partial n} = 0 \quad \text{on } \partial\Omega
\end{aligned}
$$

Here ϕ is the potential of the velocity of the fluid, c denotes the speed of sound in the fluid, ρ_0 is the specific density of the fluid, k_j represents the stiffness constant of the spring system supporting tube j, m_j is the mass per unit length of the tube j, and n is the outward unit normal on the boundary of Ω and Ω_j, respectively. Again, discretizing by finite elements yields a rational eigenproblem (1.1).

3 Minmax Characterization for Nonlinear Eigenproblems

In this section we briefly summarize the variational characterization of eigenvalues for nonlinear symmetric eigenvalue problems of finite dimension.

For $\lambda \in J$, where J is an open interval, let $T(\lambda) \in \mathbb{R}^{n\times n}$ be a symmetric matrix, whose elements are differentiable functions of λ. We assume that for every $x \in \mathbb{R}^n \setminus \{0\}$ the real equation

$$f(\lambda, x) := x^T T(\lambda) x = 0 \tag{3.5}$$

has at most one solution $\lambda \in J$. Then equation (3.5) defines a functional p on some subset $D \subset \mathbb{R}^n$ which obviously generalizes the Rayleigh quotient for linear pencils $T(\lambda) = \lambda B - A$, and which we call the Rayleigh functional of the nonlinear eigenvalue problem

$$T(\lambda)x = 0. \tag{3.6}$$

We further assume that

$$x^T T'(p(x)) x > 0 \quad \text{for every } x \in D \tag{3.7}$$

generalizing the definiteness requirement for linear pencils. By the implicit function theorem D is an open set, and differentiating the identity $x^TT(p(x))x = 0$ one obtains, that the eigenvectors of (3.6) are stationary points of p.

For overdamped problems, i.e. if the Rayleigh functional p is defined on $\mathbb{R}^n \setminus \{0\}$, Rogers [6] generalized the minmax characterization of Poincaré for symmetric eigenproblems to nonlinear ones. In this case problem (3.6) has exactly n eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ in J, and it holds

$$\lambda_j = \min_{\dim V = j} \max_{v \in V,\ v \neq 0} p(v).$$

For the general symmetric nonlinear case this characterization does not hold. This is easily seen considering a linear family $T(\lambda) = \lambda B - A$ on an interval J which does not contain the smallest eigenvalue of $Ax = \lambda Bx$.

The key idea introduced in [10] is to enumerate the eigenvalues appropriately. The value $\lambda \in J$ is an eigenvalue of problem (3.6) if and only if $\mu = 0$ is an eigenvalue of the matrix $T(\lambda)$, and by Poincaré's maxmin principle there exists $m \in \mathbb{N}$ such that

$$0 = \max_{\dim V = m} \min_{x \in V \setminus \{0\}} \frac{x^TT(\lambda)x}{\|x\|^2}.$$

Then we assign this m to λ as its number and call λ an m-th eigenvalue of problem (3.6).

With this enumeration the following minmax characterization holds (cf. [10]):

Theorem 1. *Assume that for every $x \neq 0$ equation* (3.5) *has at most one solution $p(x)$ in the open real interval J, and that condition* (3.7) *holds.*

Then for every $m \in \{1, \dots, n\}$ the nonlinear eigenproblem (3.6) *has at most one m-th eigenvalue λ_m in J, which can be characterized by*

$$\lambda_m = \min_{\substack{\dim V = m \\ D \cap V \neq \emptyset}} \sup_{v \in D \cap V} p(v). \tag{3.8}$$

Conversely, if

$$\lambda_m := \inf_{\substack{\dim V = m \\ D \cap V \neq \emptyset}} \sup_{v \in D \cap V} p(v) \in J, \tag{3.9}$$

then λ_m is an m-th eigenvalue of (3.6)*, and the characterization* (3.8) *holds.*

The minimum is attained by the invariant subspace of the matrix $T(\lambda_m)$ corresponding to its m largest eigenvalues, and the supremum is attained by any eigenvector of $T(\lambda_m)$ corresponding to $\mu = 0$.

To prove this characterization we took advantage of the following relation

$$\lambda \left\{ \begin{array}{c} > \\ = \\ < \end{array} \right\} \lambda_m \quad \Leftrightarrow \quad \mu_m(\lambda) := \max_{\dim V = m} \quad \min_{x \in V,\ x \neq 0} \quad \frac{x^TT(\lambda)x}{\|x\|^2} \left\{ \begin{array}{c} > \\ = \\ < \end{array} \right\} 0. \tag{3.10}$$

The Rayleigh functional suggests the method in Algorithm 1 called safeguarded iteration for computing the m–th eigenvalue.

The following theorem contains the convergence properties of the safeguarded iteration (cf. [11], [7]).

Algorithm 1 Safeguarded iteration

```
1: Start with an approximation $\sigma_1$ to the $m$-th eigenvalue of (3.6)
2: for $k = 1, 2, \dots$ until convergence do
3:     determine an eigenvector $x_k$ corresponding to the $m$-largest eigenvalue of $T(\sigma_k)$
4:     evaluate $\sigma_{k+1} = p(x_k)$, i.e. solve $x_k^T T(\sigma_{k+1}) x_k = 0$ for $\sigma_{k+1}$
5: end for
```

Theorem 2. *(i) If $\lambda_1 := \inf_{x \in D} p(x) \in J$ and $\sigma_1 \in p(D)$ then the safeguarded iteration converges globally to λ_1.*

(ii) If $\lambda_m \in J$ is an m-th eigenvalue of (3.6) *which is simple then the safeguarded iteration converges locally and quadratically to λ_m.*

(iii) Let $T(\lambda)$ be twice continuously differentiable, and assume that $T'(\lambda)$ is positive definite for $\lambda \in J$. If x_k in step 3. of Algorithm 1 is chosen to be an eigenvector corresponding to the m largest eigenvalue of the generalized eigenproblem $T(\sigma_k)x = \mu T'(\sigma_k)x$ then the convergence is even cubic.

4 Order Reduction for Rational Eigenproblems

Let $K \in \mathbb{R}^{N \times N}$ and $M \in \mathbb{R}^{N \times N}$ be sparse symmetric matrices where M is positive definite and K is positive semidefinite, let $C \in \mathbb{R}^{N \times k}$ be a rectangular matrix of low rank $k \ll N$, and let $D(\lambda) := \operatorname{diag}\{\frac{\lambda}{\kappa_j - \lambda m_j}\} \in \mathbb{R}^{k \times k}$ be a real diagonal matrix depending rationally on a real parameter λ.

We consider the rational eigenvalue problem

$$T(\lambda)x := -Kx + \lambda Mx + CD(\lambda)C^T x = 0. \tag{4.11}$$

Decomposing $x = Cy + z$ with $y \in \mathbb{R}^k$ and $z \in \text{range}\{C\}^\perp$, and multiplying equation (4.11) by $C^T(-K + \lambda M)^{-1}$ one obtains

$$C^T(Cy + z) + C^T(-K + \lambda M)^{-1} CD(\lambda)C^T(Cy + z) = 0$$

which is equivalent to

$$\tilde{T}(\lambda)\tilde{x} := -D(\lambda)^{-1}\tilde{x} + C^T(K - \lambda M)^{-1}C\tilde{x} = 0, \quad \tilde{x} := D(\lambda)C^T Cy. \tag{4.12}$$

This eigenproblem is of much smaller dimension than problem (4.11), and it retains the symmetry properties of (4.11). It is easily seen that $\tilde{T}(\lambda)$ satisfies the conditions of the minmax characterization in each interval $J := (\pi_j, \pi_{j+1})$ where π_j denotes the eigenvalues of the generalized problem $Kx = \pi Mx$ in ascending order. Hence, (4.12) could be solved by safeguarded iteration. However, since the dimension N of the original problem is usually very large, it is very costly to evaluate $C^T(K - \lambda M)^{-1}C$ and therefore $\tilde{T}(\lambda)$ for some given λ.

The term $H(\lambda) := C^T(K - \lambda M)^{-1}C$ appears in transfer functions of time invariant linear systems, and in systems theory techniques have been developed to reduce the order of this term considerably. One way to define such a reduction is by means of Padé approximation of $H(\lambda)$, which is a rational matrix function of the same type with a much smaller order than N.

Let $\lambda_0 \in \mathbb{C}$ be a shift which is not a pole of H. Then H has a Taylor series about λ_0

$$H(\lambda) = \sum_{j=0}^{\infty} \mu_j (\lambda - \lambda_0)^j \tag{4.13}$$

where the moments μ_j are $k \times k$ matrices. A reduced-order model of state-space dimension n is called an n-th Padé model of system (4.13), if the Taylor expansions of the transfer function H of the original problem and H_n of the reduced model agree in as many leading terms as possible, i.e.

$$H(\lambda) = H_n(\lambda) + \mathcal{O}\big((\lambda - \lambda_0)^{q(n)}\big), \tag{4.14}$$

where $q(n)$ is as large as possible, and which was proved by Freund [5] to satisfy

$$q(n) \geq 2\lfloor \frac{n}{k} \rfloor.$$

Although the Padé approximation is determined via a local property (4.14) it usually has excellent approximation properties in large domains which may even contain poles. As introduced by Feldmann and Freund [4] the Padé approximation H_n can be evaluated via the Lanczos process.

To apply the Padé–by–Lanczos process to the rational eigenproblem we transform $\tilde{T}$ further to a more convenient form. Choosing a shift λ_0 close to the eigenvalues we are interested in problem (4.12) can be rewritten as

$$\tilde{T}(\lambda)\tilde{x} = -\frac{1}{\lambda} D_1 \tilde{x} + D_2 \tilde{x} + H_{\lambda_0} \tilde{x} + (\lambda - \lambda_0) B^T (I - (\lambda - \lambda_0) A)^{-1} B \tilde{x} = 0 \tag{4.15}$$

where $M = EE^T$ is the Cholesky factorization of M, $H_{\lambda_0} := C^T(K - \lambda_0 M)^{-1} C$, $B := E^T(K - \lambda_0 M)^{-1} C$, $A := E^T(K - \lambda_0 M)^{-1} E$, and D_1 and D_2 are diagonal matrices with positive entries κ_j and m_j, respectively.

If no deflation is necessary the order of $B^T(I - (\lambda - \lambda_0)A)B$ can be reduced by block Lanczos method, and the following theorem holds.

Theorem 3. *Let $V_m \in \mathbb{R}^{N \times mk}$ be an orthonormal basis of the block Krylov space $\mathcal{K}_m(A, B) := span\{B, AB, \ldots, A^{m-1}B\}$ generated by the block Lanczos process such that the following recursion holds*

$$AV_m = V_m A_m + [O, \ldots, O, \hat{V}_{m+1} \beta_{m+1}] \tag{4.16}$$

where $\hat{V}_{m+1} \in \mathbb{R}^{N \times k}$, $\beta_{m+1} \in \mathbb{R}^{k \times k}$, and $A_m \in \mathbb{R}^{mk \times mk}$.

Then with $B = V_1 \Phi$, and $B_m := [I_k, O, \ldots, O]^T \Phi$ the moments are given by

$$BA^i B = B_m^T A_m^i B_m, \quad i = 0, 1, \ldots, 2m - 1, \tag{4.17}$$

and it holds

$$B^T (I - sA)^{-1} B = B_m^T (I - sA_m)^{-1} B_m + \mathcal{O}(|s|^{2m}). \tag{4.18}$$

A more general version taking into account deflation is proved in [5], a different approach based on a coupled recurrence is derived in [1]. Note that we will consider only real shifts and therefore all appearing matrices can be assumed to be real.

Replacing $B^T(I-(\lambda-\lambda_0)A)^{-1}B$ by $B_m^T(I-(\lambda-\lambda_0)A_m)^{-1}B_m$ one obtains the reduced eigenvalue problem

$$S(\lambda)\tilde{x} := -\frac{1}{\lambda}D_1\tilde{x}+D_2\tilde{x}+H_{\lambda_0}\tilde{x}+(\lambda-\lambda_0)B_m^T(I-(\lambda-\lambda_0)A_m)^{-1}B_m\tilde{x}=0 \quad (4.19)$$

which again is a rational eigenproblem with poles $\tilde{\pi}_0 = 0$ and $\tilde{\pi}_j = \lambda_0 + 1/\alpha_j$ where α_j, $j = 1, \dots, km$ denote the eigenvalues of A_m.

Let $\tilde{\pi}_\nu < \tilde{\pi}_{\nu+1}$ denote two consecutive poles of S, and let $J_\nu = (\tilde{\pi}_\nu, \tilde{\pi}_{nu+1})$. Then for $\lambda \in J_\nu$ it holds

$$\tilde{x}^T S'(\lambda)\tilde{x} = \frac{1}{\lambda^2}\tilde{x}^T D_1\tilde{x} + \sum_{j=1}^{km}\frac{\beta_j^2}{(1-\alpha_j(\lambda-\lambda_0))^2} > 0, \; B_m\tilde{x} = (\beta_j)_{j=1,\dots,km},$$

and hence the conditions of the minmax characterization are satisfied for the reduced eigenproblem in every interval J_ν, and therefore its eigenvalues can be determined by safeguarded iteration.

Moreover, for every $\tilde{x}$ the Rayleigh quotient of $S(\lambda)$

$$R(\tilde{x};\lambda) := \frac{\tilde{x}^T S(\lambda)\tilde{x}}{\|\tilde{x}\|_2^2}$$

is monotonely increasing in J_ν with respect to λ. Hence, if $\mu_j(\lambda)$ denote the eigenvalues of $S(\lambda)$ ordered in decreasing order, then every $\mu_j(\lambda)$ is monotonely increasing, and it follows immediately from (3.10)

Theorem 4. *For two consecutive poles $\tilde{\pi}_\nu < \tilde{\pi}_{\nu+1}$ of $S(\cdot)$, and $\tilde{\pi}_\nu < \xi < \zeta < \tilde{\pi}_{nu+1}$ let*

$$\mu_{\ell_1}(\xi) \le 0 < \mu_{\ell_1-1} \quad \textit{and} \quad \mu_{\ell_2}(\zeta) < 0 \le \mu_{\ell_2-1}(\zeta).$$

Then the reduced eigenvalue problem (4.19) *has $\ell_2 - \ell_1$ eigenvalues*

$$\xi \le \lambda_{\ell_1} \le \lambda_{\ell_1+1} \le \dots \le \lambda_{\ell_2-1} \le \zeta$$

which can be determined by (the cubically convergent version of) safeguarded iteration.

5 Numerical Results

We consider a clamped plate occupying the region $\Omega = (0,4) \times (0,3)$, and we assume that 6 identical masses are attached at the points (i,j), $i = 1,2,3$, $j = 1,2$.

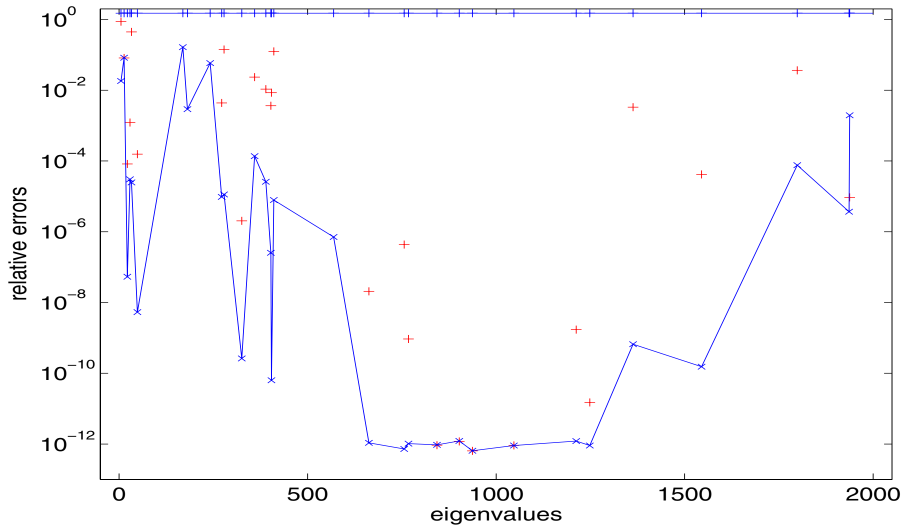
Discretizing with Bogner–Fox–Schmid elements with stepsize $h = 0.05$ one gets a rational eigenvalue problem

$$T(\lambda)x := -Kx + \lambda Mx + \frac{1000\lambda}{1000-\lambda}C^T Cx = 0$$

of dimension $N = 18644$ and $k = 6$ governing free vibrations of the plate which has 32 eigenvalues in the interval $J = (0, 2000)$.

For $m = 12$ with shift $\lambda_0 = 1000$ all 32 eigenvalues are found requiring 103.5 seconds on an Intel Centrino M processor with 1.7 GHz and 1 GB RAM under MATLAB 6.5. For $m = 6$ only 27 eigenvalues are found in 50.8 sec. For comparison, the nonlinear Arnoldi in [9] requires 227.1 seconds

Figure 1 demonstrates the approximation properties of the reduced problem. The eigenvalues are marked as vertical bars at the top of the picture, crosses indicate the relative errors of the eigenvalue approximations obtained for $m = 12$, and plus signs the errors for 27 eigenvalue approximations obtained for $m = 6$.



References

1. Z. Bai and R.W. Freund. A symmetric band Lanczos process based on coupled recurrences and some applications. *SIAM J. Sci. Comput.*, 23:542 – 562, 2001.
2. T. Betcke and H. Voss. A Jacobi–Davidson–type projection method for nonlinear eigenvalue problems. *Future Generation Computer Systems*, 20(3):363 – 372, 2004.
3. C. Conca, J. Planchard, and M. Vanninathan. *Fluid and Periodic Structures*, volume 24 of *Research in Applied Mathematics*. Masson, Paris, 1995.
4. P. Feldmann and R.W. Freund. Efficient linear circuit analysis by Padé approximation via the Lanczos process. In *Proceedings of EURO-DAC '94 with EURO-VHDL '94*, pages 170 – 175, Los Alamitos, Ca., 1994. IEEE Computer Society Press.
5. R.W. Freund. Computation of matrix Padé approximations of transfer functions via a Lanczos-type process. In C. Chui and L. Schumaker, editors, *Approximation Theory VIII, Vol. 1: Approximation and Interpolation*, pages 215 – 222, Singapore, 1995. World Scientific Publishing Co.
6. E.H. Rogers. A minmax theory for overdamped systems. *Arch.Rat.Mech.Anal.*, 16:89 – 96, 1964.

7. H. Voss. Initializing iterative projection methods for rational symmetric eigenproblems. In *Online Proceedings of the Dagstuhl Seminar Theoretical and Computational Aspects of Matrix Algorithms, Schloss Dagstuhl 2003*, `ftp://ftp.dagstuhl.de/pub/Proceedings/03/03421/03421.VoszHeinrich.Other.pdf`, 2003.
8. H. Voss. A rational spectral problem in fluid–solid vibration. *Electronic Transactions on Numerical Analysis*, 16:94 – 106, 2003.
9. H. Voss. An Arnoldi method for nonlinear eigenvalue problems. *BIT Numerical Mathematics*, 44:387 – 401, 2004.
10. H. Voss and B. Werner. A minimax principle for nonlinear eigenvalue problems with applications to nonoverdamped systems. *Math.Meth.Appl.Sci.*, 4:415–424, 1982.
11. H. Voss and B. Werner. Solving sparse nonlinear eigenvalue problems. Technical Report 82/4, Inst. f. Angew. Mathematik, Universität Hamburg, 1982.

Parallel Global Optimization of Foundation Schemes in Civil Engineering

Raimondas Čiegis, Milda Baravykaitė, and Rimantas Belevičius

Vilnius Gediminas Technical University
Saulėtekio Str. 11, LT-10223 Vilnius, Lithuania
{rc,mmb,rb}@fm.vtu.lt

Abstract. The important problem in civil engineering is obtaining optimal pile placement schemes for grillage-type foundations. The standard technique for solution of problem is application of local optimization methods, starting from initial guess obtained by means of engineering heuristics. We propose a new technology based on the use of global optimization algorithms, which are implemented as the "black-boxes". In the real world applications the problems of interest involve till 500 design parameters. Therefore such problems can not be solved using one processor due to the huge CPU and computer memory requirements. We propose a parallel version of global optimization algorithm. The results of computational experiments for a number of practical problems are presented. Efficiency of suggested parallel algorithms is investigated and results for different clusters of workstations are presented.

1 Introduction

Optimization is an inherent part of all engineering practice. In the construction of buildings that means, all parts of a building from foundations to the roof should be designed and built optimally and thrifty as much as the conditions of safety and comfort allow. In this paper we shall concentrate on the optimal design of grillage-type foundations, which are the most popular and effective scheme of foundations, especially in case of weak grounds.

Grillage consists of separate beams, which may be supported by piles, or may reside on other beams, or a mixed scheme may be the case. The optimal placement of grillage should possess, depending on given carrying capacities of piles, the minimum possible number of piles. Theoretically, reactive forces in all piles should approach the limit magnitudes of reactions for those piles. Limit pile reactions may differ from beam to beam provided different characteristics (i.e., diameters, lengths, profiles) of piles are given. Practically, this is difficult to achieve for grillages of complex geometries, and it is even a more difficult task if a designer due to some considerations introduces into the grillage scheme the so-called *immovable supports*. Such supports should retain their positions and do not participate in optimization process. These goals can be achieved by choosing appropriate pile positions.

A designer can obtain the optimal pile placement schema by implementing the well known engineering tests algorithms, which are described e.g. in [2]. However this is likely only in case of simple geometries, simple loadings and limited number of design parameters. The works employing local optimization methods were published by Baravykaite et al. [1], Belevičius et al. [2], Kim et al. [7]. The proposed techniques enabled the authors to find only local minimum solutions. But it is evident that the global minimum solution is desirable and it can improve the quality of obtained grillage scheme.

The main goal of this work is to use global optimization techniques in order to find optimal placement of piles. Here we should take into account the fact that the objective function is not given analytically and the properties of this function (e.g. the Lipschitz constant) can not be obtained apriori. The value of the objective function for fixed values of its arguments is obtained by solving a nonlinear PDE problem. Thus we should use so called *black box* global minimization algorithms.

The size of the given problem consists of hundreds of design parameters, thus we have developed a parallel version of the algorithm. Efficiency of the obtained algorithm is investigated using different clusters of workstations.

2 Local Optimization

A full formulation of the mathematical model for determination of reactive forces in piles is given in [2].

2.1 Idealization

It is well-known that minimization problem of reactive forces in piles is non-linear and non-convex. The following algorithm was employed for solution of this problem in [2].

The optimization of a single beam is the basis for whole optimization process of grillage. A grillage is divided into separate beams, the *upper* beams are resting on the *lower* beams. First, all beams are analyzed and optimized separately. Joints and intersections of the upper and lower beams are idealized as immovable supports for upper beams. Reactive forces in these fictitious supports are obtained during the analysis stage of the upper beams. Joints for the lower beams are idealized as concentrated loads of same magnitude but of opposite sign as reactive forces in fictitious supports. If more than two beams meet at joint, all beams are considered to be the "uppers" except for one – the "lower". Distinguishing between the upper and lower beams can be done automatically by program or by the designer. Later on, as these fictitious reactions/concentrated loads depend on pile positions obtained in optimization problem, all calculations are iterated in order to level with proper accuracy forces at joints (or stiffnesses, if desired). The problem has to be solved in static and in linear stage. Now let us consider optimization of a particular beam of grillage.

2.2 Mathematical Model

The optimization problem is stated as follows (see also our paper [1])

$$\min_{\text{s.t. } \mathbf{x} \in D} P(\mathbf{x}), \tag{2.1}$$

where P is the objective function, D is the feasible shape of structure, which is defined by the type of certain supports, the given number and layout of different cross-sections, and different materials in the structure, and x are design parameters.

In our problem P is defined by the maximum difference between vertical reactive force at a support and allowable reaction for this support, thus allowing us to achieve different reactions at supports on different beams, or even at particular supports on the same beam:

$$P(\mathbf{x}) = \max_{1 \le i \le N_s} \left| R_i - f_i \, R_{allowable} \right| ,$$

here N_s denotes the number of supports, $R_{allowable}$ is allowable reaction, f_i are factors to this reaction and R_i are reactive forces in each support and x are nodal co-ordinates of all (or a chosen set of) supports.

Further, the minimum–maximum problem is converted to a pure minimum problem with constraints by treating P_{max} as unknown subject to constraints that P_{max} limits the magnitudes of P everywhere in the structure and for all load cases when design changes $\varDelta x_i$ are performed:

$$P(\mathbf{x}) + \sum_{i=1}^{N_s} \frac{\partial P(\mathbf{x})}{\partial x_i} \varDelta x_i - P_{max} \le 0 \,. \tag{2.2}$$

All derivatives $\dfrac{\partial P(\mathbf{x})}{\partial x_i}$ are computed numerically by using finite difference approximations of the second order accuracy.

For computational reasons a beam length constraint is also included into formulation of the problem:

$$L(\mathbf{x}) + \sum_{i=1}^{N_s} \frac{\partial L(\mathbf{x})}{\partial x_i} \varDelta x_i - L_0 \le 0 \,, \tag{2.3}$$

where L_0 is the initial length of the model.

Several possibilities exist in the choice of design parameters x_i. Our choice is to use the most evident from the engineering point of view parameters: nodal co-ordinates of all (or a chosen set of) supports.

The problem is strongly non-linear, thus it is solved iteratively. In each iteration the current shape is changed to a better neighbouring shape. The solution requires three steps:

- finite element analysis,
- sensitivity analysis with regard to design parameters,
- optimal re-design with linear programming.

Simple two-node beam element with 4 d.o.f.'s has been implemented in analysis, therefore the analytical sensitivity analysis with regard to nodal co-ordinates was feasible. "Move limit technique" (see, Pedersen [9]) relates the design parameters' alterations per one iteration of Simplex procedure to the objective function and assures adjustment of those alterations to the extent of problem non-linearity (see, e.g., [2] for the details).

2.3 Optimization Algorithm

Optimization of separate beams has to be included into a general iterative algorithm for optimization of the whole system of piles, because pile placement scheme of one beam influences reactions distribution in remaining beams. The following algorithm is employed:

```
Algorithm 1
Initialization:
        Set stiffnesses at fictitious immovable supports of upper
        beams simulating joints with lower beams and
        accuracy tolerance.
        Set $stop \leftarrow false$.
while ( $stop = false$ ) do
        1.1. Optimize the upper beams using defined in the last
             iteration stiffnesses of fictitious immovable supports.
        1.2. Optimize the lower beams in addition to specified
             loadings taking into account concentrated loads
             coming from the upper beams.
        1.3. if ( stiffnesses of the upper and lower beams at joints
                 match (with specified accuracy) ) do
                 Set $stop \leftarrow true$.
             end if
end while
1.4. Filtering results to exclude matching supports at joints of beams.
The Simplex method is used to solve optimization subproblems in steps 1.1 and 1.2.
```

In order to find a proper solution, the algorithm should be launched from a near-optimum initial scheme, which is delivered by special expert system. This should ensure the global solution for many practical foundation schemes. However, it was shown that grillages expose high sensitivity to the pile positions, especially when the stiffness characteristics of piles are not known exactly and the piles have to be idealized as the rigid supports. In all those cases solutions are evidently far from global ones.

3 Global Optimization

The most simple global optimization algorithm is obtained by starting local search algorithm from many different trial points. Such strategy can be easily implemented in parallel (see, [1]). In some cases it gives sufficiently good solutions. The quality of the obtained solution depends on the initial trial points, and some heuristic for the selection of such initial approximations should be given (see [2]).

In general the probability of finding the global minimum can be increased by using a big number of starting points of local searches and it approaches one only when the number of trial points approaches the infinity (or when we have some apriori information about the distribution of possible solutions).

4 Branch and Bound Algorithm

In our work we have applied more sophisticated methods, which are based on branch and bound (*BB*) algorithms. For an introduction to BB method we refer the reader to [6].

Let assume that we solve a global minimization problem. Any BB algorithm consists of two main steps: a) branching rule, i.e. we select a subdomain from D and divide it into two or more smaller parts, b) computation of lower bounds on the objective function value for each new subdomain. If the lower bound of the objective function is lager than the best known approximation of the value of this function, then such a subdomain it is taken out from farther searches.

The starting point of our method is the algorithm developed by Žilinskas [11]. This method can be interpreted as a strategy for managing local searches in a search for global minimizers. Once a local minimizer has been found by the local search method described above, a domain around it is taken out from farther searches. The branch strategy is taken from [11].

Main Steps of the BB Algorithm

- The monotonicity test, which tries to prove that no local minimum exists in the tested subdomain;
- Once a local minimizer has been found, a domain around it is taken out from father searches;
- A subdivision step, where the estimation of the lower bound is computed, if needed.

We note that the lower bound of the objective function can be computed only approximately, since the objective function is not given explicitly. Thus we can not guarantee that the exact global minimum solution is obtained. But in real world applications we are mostly interested in finding a sufficiently good solution, since the problem is too large to be solved exactly. The stopping condition of computations is usually specified by a limit of CPU time. Thus parallel computations are aimed to test a bigger subset of D, but not to decrease a time of computations.

5 Parallel Algorithm

Parallelization of branch and bound type algorithms is a difficult problem. It has many common points with parallel adaptive multidimensional integration problem (see a review of such methods in Čiegis et al. [3]). The most obvious way is to use a functional parallelization paradigm, for example when the objective function is computed in parallel. One interesting approach for parallel global minimization is proposed by Groenwold and Hindley [5], who have applied competing in parallel different algorithms for solving a problem of structural optimization.

The parallel implementation of the branch and bound type global minimization algorithm from [11] was obtained in [12] by using the domain distribution method. The domain is divided into sub-domains, which are distributed among processors. Then all processors work independently using the sequential algorithm and exchanging between processors only the currently known best value of the objective function. The efficiency of this type parallel algorithms depends on the quality of the apriori sub-problems distribution.

In most cases some processors (or even a big part of processors) will become idle. Thus we have applied a distributed dynamic load balancing algorithm from [3] to ensure that the work load is distributed uniformly (see, also [4,8,10], where different parallel branch and bound methods are given and investigated).

Application of the dynamic load balancing algorithm improves the load balance among processors but introduces additional synchronization and data movement costs. Thus some balance between decrease of computation time and enlargement of data sending time should be preserved.

But even if a time used for communications between processors is neglected, the efficiency of the parallel algorithm can be not good. It is due to the fact, that branching strategy of the parallel *BB* algorithm is different from the serial version of *BB* algorithm. For example, if the serial *BB* algorithm uses the *depth first search* branching strategy and the objective function has many similar local minimum points, then it is possible that all work, which is done by the other processors, will be useless.

6 Numerical Examples

To illustrate the proposed technology, two support placement schemes for relatively simple grillages were obtained. Our codes require only input on geometry of grillage, loading, grillage material characteristics, and on support characteristics (i.e. carrying capacity or stiffnesses). Also, some scalars render limitations on scheme: the allowable vertical reaction, minimum allowable distance between two adjacent support due to technological reasons of construction, maximum allowable deflection of console-parts of grillage should be given.

The optimization program-kernel is embraced with design program *Matrix Frame*, which supplies all the pre- and postprocessors needed. It should be noted, some grillage schemes expose extreme sensitivity to the positions of supports. Small changes of supports' co-ordinates may raise significant perturbations in reactive forces. The two examples below belong just to this category of schemes. Earlier, using local optimization methods, we even did not obtain a reasonable solutions for these problems. As to the results of examples, there the global solution, i.e., an even distribution of reactive forces across the scheme was not achieved, because the design program stops optimization after the allowable reaction is reached.

Computational experiments are performed on a cluster of Vilnius Gediminas technical university. It consist of 10 dual processor PC, running LINUX.

Example 1. Grillage of rectangular shape loaded with two sets of distributed vertical loadings (Fig. 1a). Construction of grillage consists of standard prefab reinforced concrete girders. The main determinant data for support scheme are the maximum allowable vertical reaction, the minimum allowable distance between two adjacent supports, and the vertical stiffness of support: 200, 0.210, and 1.e15, accordingly. Theoretical number of supports is 10.

As we have mentioned above, the stopping condition of computations is specified by a limit of CPU time. After 60 minutes of computations the proposed algorithm yields the

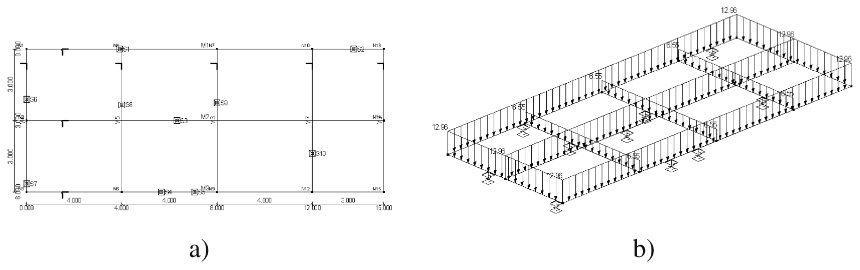


Fig. 1. a) geometry, b) loadings and the obtained optimized pile placement scheme

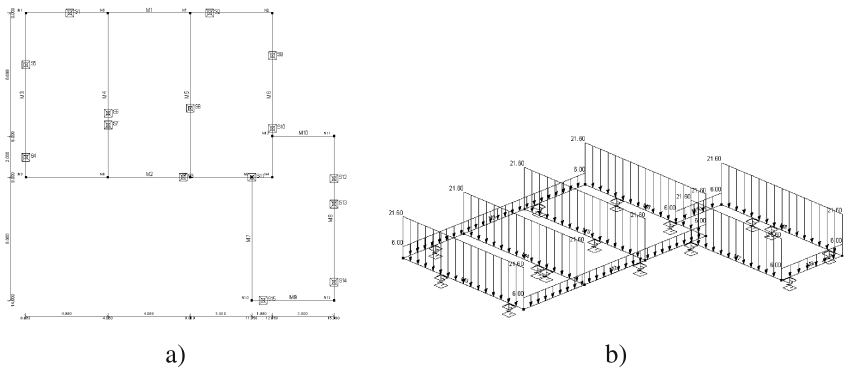


Fig. 2. a) geometry, b) loadings and the obtained optimized pile placement scheme

following supports placement scheme (see Fig. 1b), for which the maximum allowable reaction (with given tolerance of 0.05) 207.5 is achieved. All reactive forces for supports, ordered as shown in Fig. 1, are the following:

$$
\begin{aligned}
&-206.3,\ -170.6,\ -204.5,\ -189.0,\ -94.96,\\
&-196.8,\ -168.6,\ -207.5,\ -192.1,\ -207.4\,.
\end{aligned}
$$

Example 2. Grillage consists of two rectangular frames under distributed loadings (see Fig. 2a). Theoretical number of supports for initial data on limiting factors 150, 0.10, 1.e10 (as in the Example 1) is 15. After 5 hours of computations on 20 processors cluster the following placement scheme was achieved (see Fig. 2b). The obtained reactions are the following: -156.4, -158.1, -147.4, -161.2, -153.6, -118.8, -130.6, -161.3, -161.5, -139.7, -161.2, -144.0, -94.37, -121.1, -136.4.

Acknowledgements

The authors would like to thank one of the referee for his constructive criticism which helped to improve the clarity of this note.

This work was partially supported by the Lithuanian State Science and Studies Foundation within the framework of the Eureka project OPTPAPER EU-2623, E-2002.02.27.

References

1. M. Baravykaite, R. Belevičius, and R. Čiegis. One application of the parallelization tool of Master – Slave algorithms. *Informatica*, 13(4): 393–404, 2002.
2. R. Belevičius, S. Valentinavičius, and E. Michnevič. Multilevel optimization of grillages. *Journal of Civil Engineering and Management*, 8(1): 98–103, 2002.
3. R. Čiegis, R. Šablinskas, and J. Wasniewski. Hyper-rectangle distribution algorithm for parallel multidimensional numerical integration. In J. Dongarra, E. Luque, and T. Margalef, editors, *Proceedings of 6th European PVM/MPI User's Group Meeting, Recent advances in PVM and MPI*, in Lecture Notes in Computer Science, Number 1697, pages 275–282, Barselona, Spain, June 1999. Springer.
4. J. Clausen. Parallel search–based methods in optimization. *Applied Parallel Computing – Industrial Computation and Optimization, PARA96*, in Lecture Notes in Computer Science, Number 1184, pages 176–185, 1997. Springer.
5. A.A. Groenwold and M.P. Hindley. Competing parallel algorithms in structural optimization. *Struct. Multidisc. Optim.*, 24: 343–350, 2002.
6. R. Horst and H.Tuy. *Global Optimization: Deterministic Approaches.* Springer–Verlag, 1993.
7. K. Kim, S. Lee, C. Chung, and H. Lee. Optimal pile placement for minimizing differential settlements in piled raft foundations.
   http://strana.snu.ac.kr/laboratory/publications, 2004.
8. V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing*. The Benjamin/Cummings Publishing Company, Inc., Amsterdam, Bonn, Tokyo, Madrid, 1994.
9. P. Pedersen. Design for minimum stress concentration – some practical aspects. In: *Structural Optimization*, Kluwer Academic, 225–232, 1989.
10. Ch. Xu and F. Lau. *Load balancing in parallel computers.* Kluwer Academic Publishers, Boston, Dordrecht, London, 1997.
11. J. Žilinskas. Black Box global optimization inspired by interval methods. *Information Technology and Control*, 21(4): 53–60, 2001.
12. J. Žilinskas. *Black box global optimization: covering methods and their parallelization.* Doctoral dissertation, Kaunas Technological University, Kaunas, 2002.

A Combined Linear and Nonlinear Preconditioning Technique for Incompressible Navier-Stokes Equations*

Feng-Nan Hwang and Xiao-Chuan Cai

Department of Computer Science, University of Colorado, Boulder, CO 80309, USA
hwangf@colorado.edu, cai@cs.colorado.edu

Abstract. We propose a new two-level nonlinear additive Schwarz preconditioned inexact Newton algorithm (ASPIN). The two-level nonlinear preconditioner combines a local nonlinear additive Schwarz preconditioner and a global *linear* coarse preconditioner. Our parallel numerical results based on a lid-driven cavity incompressible flow problem show that the new two-level ASPIN is nearly scalable with respect to the number of processors if the coarse mesh size is fine enough.

1 Introduction

We focus on the parallel numerical solution of large, sparse nonlinear systems of equations arising from the finite element discretization of nonlinear partial differential equations. Such systems appear in many computational science and engineering applications, such as the simulation of fluid flows [8]. In particular, we introduce a nonlinearly preconditioned iterative method that is robust and scalable for solving nonlinear systems of equations. Our approach is based on the inexact Newton method with backtracking technique (INB) [4], which can be briefly described as follows. Let

$$F(x^*) = 0 \tag{1.1}$$

be a nonlinear system of equations and $x^{(0)}$ a given initial guess. Assume $x^{(k)}$ is the current approximate solution. Then a new approximate solution $x^{(k+1)}$ of (1.1) can be computed by first finding an inexact Newton direction $s^{(k)}$ satisfying

$$||F(x^{(k)}) + F'(x^{(k)})s^{(k)}||_2 \leq \eta_k ||F(x^{(k)})||_2,$$

then obtaining $x^{(k+1)}$ with $x^{(k+1)} = x^{(k)} + \lambda^{(k)} s^{(k)}$. The scalar η_k is often called the "forcing term", which determines how accurately the Jacobian system needs to be solved by some iterative method, such as GMRES. The scalar $\lambda^{(k)}$ is selected using a linesearch technique. Although INB has the desirable property of local fast convergence, like other nonlinear iterative methods, INB is very fragile. It converges rapidly for a well-selected set of parameters (for example, certain initial guess, certain range of the Reynolds number Re), but fails to converge due to a change in some parameters. It is

* The work was partially supported by the Department of Energy, DE-FC02-01ER25479, and by the National Science Foundation, CCR-0219190, ACI-0072089 and CCF-0305666.

often observed that INB converges well at the beginning of the iterations, then suddenly stalls for no apparent reason. In [2,3,6] some nonlinear preconditioning techniques were developed, and the convergence of Newton-type methods becomes not sensitive to these unfriendly parameters if INB is applied to a nonlinearly preconditioned system

$$\mathcal{F}(x^*) = 0 \tag{1.2}$$

instead. Here the word “preconditioner” refers to the fact that systems (1.1) and (1.2) have the same solution and the new system (1.2) is better conditioned, both linearly and nonlinearly. The preconditioner is constructed using a nonlinear additive Schwarz method. To improve the processor scalability, a two-level method was then proposed in [3], which works well if the number of processors is not large. For a large number of processors, the *nonlinear* coarse solver takes too much CPU and communication times. In this paper, we suggest a combined *linear and nonlinear* additive Schwarz preconditioner and show that using a linear coarse solver we can retain the nonlinear robustness and reduce the nonlinear complexity considerably.

2 Nonlinear Preconditioning Algorithms

In this section, we describe a two-level nonlinear preconditioner based on a combination of local nonlinear additive Schwarz preconditioners and a global linear coarse preconditioner. We restrict our discussion to a two-component system (velocity and pressure) resulting from the finite element discretization of two-dimensional steady-state incompressible Navier-Stokes equations defined on a bounded domain Ω in R^2 with a polygonal boundary Γ:

$$\begin{cases} \boldsymbol{u} \cdot \nabla \boldsymbol{u} - 2\nu \nabla \cdot \epsilon(\boldsymbol{u}) + \nabla p = 0 & \text{in} \quad \Omega, \\ \nabla \cdot \boldsymbol{u} = 0 & \text{in} \quad \Omega, \\ \boldsymbol{u} = \boldsymbol{g} & \text{on} \quad \Gamma, \end{cases} \tag{2.3}$$

where $\boldsymbol{u} = (u_1, u_2)$ is the velocity, p is the pressure, $\nu = 1/Re$ is the dynamic viscosity, and $\epsilon(\boldsymbol{u}) = 1/2(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T)$ is the symmetric part of the velocity gradient. The pressure p is determined up to a constant. To make p unique, we impose an additional condition $\int_\Omega p\, dx = 0$. To discretize (2.3), we use a stabilized finite element method [5] on a given quadrilateral mesh $\mathcal{T}^h = \{K\}$. Let V^h and P^h be a pair of finite element spaces for the velocity and pressure, given by

$$V^h = \{\boldsymbol{v}^h \in (C^0(\Omega) \cap H^1(\Omega))^2 : \boldsymbol{v}|_K \in Q_1(K)^2,\ K \in \mathcal{T}^h\} \text{ and}$$
$$P^h = \{p^h \in C^0(\Omega) \cap L^2(\Omega) : p|_K \in Q_1(K),\ K \in \mathcal{T}^h\}.$$

Here, $C^0(\Omega)$, $L^2(\Omega)$, and $H^1(\Omega)$ are the standard notations with usual meanings in the finite element literature [5]. For simplicity, our implementation uses a $Q_1 - Q_1$ element (continuous bilinear velocity and pressure). The weighting and trial velocity function spaces V_0^h and V_g^h are

$$V_0^h = \{\boldsymbol{v}^h \in V^h : \boldsymbol{v} = \boldsymbol{0} \text{ on } \Gamma\} \text{ and } V_g^h = \{\boldsymbol{v}^h \in V^h : \boldsymbol{v} = \boldsymbol{g} \text{ on } \Gamma\}.$$

Similarly, let the finite element space P_0^h be both the weighting and trial pressure function spaces:

$$P_0^h = \{p^h \in P^h : \int_\Omega p\, dx = 0\}.$$

Following [5], the stabilized finite element method for steady-state incompressible Navier-Stokes equations reads: Find $\boldsymbol{u}^h \in V_g^h$ and $p^h \in P_0^h$, such that

$$B(\boldsymbol{u}^h, p^h; \boldsymbol{v}, q) = 0 \qquad\qquad \forall(\boldsymbol{v}, q) \in V_0^h \times P_0^h \tag{2.4}$$

with

$$\begin{aligned}
&B(\boldsymbol{u}, p; \boldsymbol{v}, q) = \\
&((\nabla \boldsymbol{u}) \cdot \boldsymbol{u}, \boldsymbol{v}) + (2\nu\epsilon(\boldsymbol{u}), \epsilon(\boldsymbol{v})) - (\nabla \cdot \boldsymbol{v}, p) - (\nabla \cdot \boldsymbol{u}, q) + (\nabla \cdot \boldsymbol{u}, \delta \nabla \cdot \boldsymbol{v}) + \\
&\sum_{K \in \mathcal{T}^h} ((\nabla \boldsymbol{u}) \cdot \boldsymbol{u} + \nabla p - 2\nu \nabla \cdot \epsilon(\boldsymbol{u}), \tau((\nabla \boldsymbol{v}) \cdot \boldsymbol{v} - \nabla q - 2\nu \nabla \cdot \epsilon(\boldsymbol{u}))_K.
\end{aligned}$$

We use the stabilization parameters δ and τ as suggested in [5]. The stabilized finite element formulation (2.4) can be written as a nonlinear algebraic system

$$F(x) = 0, \tag{2.5}$$

which is often large, sparse, and highly nonlinear when the value of Reynolds number is large. The vector x corresponds to the nodal values of $\boldsymbol{u}^h = (u_1^h, u_2^h)$ and p^h in (2.4).

2.1 Subdomain Partition and One-Level Nonlinear Preconditioner

To define parallel Schwarz type preconditioners, we partition the finite element mesh $\mathcal{T}^h$ introduced in the previous section. Let $\{\Omega_i^h, i = 1, ..., N\}$ be a non-overlapping subdomain partition whose union covers the entire domain Ω and its mesh $\mathcal{T}^h$. We use $\mathcal{T}_i^h$ to denote the collection of mesh points in Ω_i^h. To obtain overlapping subdomains, we expand each subdomain Ω_i^h to a larger subdomain $\Omega_i^{h,\delta}$ with the boundary $\partial\Omega_i^{h,\delta}$. Here δ is an integer indicating the degree of overlap. We assume that $\partial\Omega_i^{h,\delta}$ does not cut any elements of $\mathcal{T}^h$. Similarly, we use $\mathcal{T}_i^{h,\delta}$ to denote the collection of mesh points in $\Omega_i^{h,\delta}$. Now, we define the subdomain velocity space as

$$V_i^h = \{\boldsymbol{v} \in V^h \cap (H^1(\Omega_i^{h,\delta}))^2 : \boldsymbol{v}^h = 0 \text{ on } \partial\Omega_i^{h,\delta}\}$$

and the subdomain pressure space as

$$P_i^h = \{p^h \in P^h \cap L^2(\Omega_i^{h,\delta}) : p^h = 0 \text{ on } \partial\Omega_i^{h,\delta} \backslash \Gamma\}.$$

On the physical boundaries, Dirichlet conditions are imposed according to the original equations (2.3). On the artificial boundaries, both $\boldsymbol{u} = 0$ and $p = 0$.

Let $R_i : V^h \times P^h \to V_i^h \times P_i^h$ be a restriction operator, which returns all degrees of freedom (both velocity and pressure) associated with the subspace $V_i^h \times P_i^h$. R_i is an $3n_i \times 3n$ matrix with values of either 0 or 1, where n and n_i are the total number of mesh points in $\mathcal{T}^h$ and $\mathcal{T}_i^{h,\delta}$, respectively, and $\sum_{i=1}^N 3n_i \geq 3n$. Note that for $Q_1 - Q_1$ elements, we have three variables per mesh point, two for the velocity and one for the pressure. Then, the interpolation operator R_i^T can be defined as the transpose of R_i. The multiplication of R_i (and R_i^T) with a vector does not involve any arithmetic operation, but does involve communication in a distributed memory parallel implementation. Using the restriction operator, we define the subdomain nonlinear function $F_i : R^{3n} \to R^{3n_i}$ as

$$F_i = R_i F.$$

We next define the subdomain mapping functions, which in some sense play the role of subdomain preconditioners. For any given $x \in R^{3n}$, $T_i(x) : R^{3n} \to R^{3n_i}$ is defined as the solution of the following subspace nonlinear systems,

$$F_i(x - R_i^T T_i(x)) = 0, \text{ for } = 1, ..., N. \tag{2.6}$$

Throughout this paper, we always assume that (2.6) is uniquely solvable. Using the subdomain mapping functions, we introduce a new global nonlinear function,

$$\mathcal{F}^{(1)}(x) = \sum_{i=1}^{N} R_i^T T_i(x), \tag{2.7}$$

which we refer to as the nonlinearly preconditioned $F(x)$. The one-level additive Schwarz inexact preconditioned Newton algorithm (ASPIN(1)) is defined as: Find the solution x^* of (2.5) by solving the nonlinearly preconditioned system,

$$\mathcal{F}^{(1)}(x) = 0, \tag{2.8}$$

using INB with an initial guess $x^{(0)}$. As shown in [2,6], an approximation of the Jacobian of $\mathcal{F}^{(1)}$ takes the form

$$\widehat{\mathcal{J}}^{(1)}(x) = \sum_{i=1}^{N} J_i^{-1} J(x), \tag{2.9}$$

where J is the Jacobian of the original function $F(x)$ and $J_i = R_i J R_i{}^T$.

2.2 A Parallel Linear Coarse Component for the Nonlinear Preconditioner

The one-level ASPIN is robust, but not linearly scalable with respect to the number of processors. Some coarse preconditioner is required to couple the subdomain preconditioners. One such coarse preconditioner is proposed and tested in [3,7]. The nonlinear coarse system is obtained by the discretization of original nonlinear partial differential equations on a coarse mesh. Although, in general, solving the coarse systems is easier than the fine systems, a Newton-Krylov-Schwarz method sometimes is not good enough to converge the coarse system. Therefore, ASPIN(1) is used to solve the coarse system in [3,7]. To evaluate the coarse function at certain point, one needs to solve a set of nonlinear systems of equations. Although the ASPIN(1)-based coarse solver provides good mathematical properties, such as helping speed up the convergence of the linear iterative method, the computational cost to solve many coarse systems is usually high in practice. Numerical experiments [7] show that the ASPIN(1)-based coarse solver works fine only for a moderate number of processors, for a large number of processors, a more efficient coarse solver is needed.

Here we introduce a new coarse system, which is linear, and the system is constructed by a linearization of the nonlinear coarse system mentioned above, using a Taylor approximation. The coarse function evaluation only requires the solution of a linear system, and hence the computational cost is reduced considerably. More precisely, we assume there exists a finite element mesh $\mathcal{T}^H$ covering the domain Ω. The two meshes $\mathcal{T}^H$ and $\mathcal{T}^h$ do not have to be nested. For the purpose of parallel computing, the coarse mesh is partitioned into non-overlapping subdomains $\{\Omega_i^H\}$ and overlapping subdomains $\{\Omega_i^{H,\delta}\}$. The corresponding sets of mesh points are denoted by $\{\mathcal{T}_i^H\}$, and $\{\mathcal{T}_i^{H,\delta}\}$. For the simplicity of our software implementation, we assume a non-overlapping partition to be *nested*. In other words, we must have

$$\Omega_i^h = \Omega_i^H$$

for $i = 1, \ldots, N$, even though the corresponding sets of mesh points do not have to be nested; i.e.,

$$\mathcal{T}_i^h \neq \mathcal{T}_i^H.$$

This also means that the same number of processors is used for both the fine and coarse mesh problems. If the overlap is taken into account, in general,

$$\Omega_i^{h,\delta} \neq \Omega_i^{H,\delta}, \quad \text{and} \quad \mathcal{T}_i^{h,\delta} \neq \mathcal{T}_i^{H,\delta}.$$

As in the fine mesh case, we can also define the restriction and extension operators R_i^c and $(R_i^c)^T$ for each coarse subdomain. On the coarse mesh $\mathcal{T}^H$, we can define finite element subspaces similar to the ones defined on the fine meshes, and discretize the original Navier-Stokes equations to obtain a nonlinear system of equations,

$$F^c(x_c^*) = 0, \tag{2.10}$$

where the coarse solution x_c^* of (2.10) is determined through a pre-processing step. Similar to the fine mesh, on the coarse subdomains, we obtain the coarse Jacobian submatrices

$$J_i^c = (R_i^c) J^c (R_i^c)^T, i = 1, \ldots, N,$$

where J^c is the Jacobian matrix of the coarse mesh function F^c.

We next define the coarse-to-fine and fine-to-coarse mesh transfer operators. Let $\{\phi_j^H(x), j = 1, \ldots, m\}$ be the finite element basis functions on the coarse mesh, where m is the total number of coarse mesh points in $\mathcal{T}^H$. We define an $3n \times 3m$ matrix I_H^h, the coarse-to-fine extension matrix, as

$$I_H^h = \left[E_1 \, E_2 \cdots E_n\right]^T,$$

where the block matrix E_i of size $3 \times 3m$ is given by

$$E_i = \begin{bmatrix} (e_H^h)_i & 0 & 0 \\ 0 & (e_H^h)_i & 0 \\ 0 & 0 & (e_H^h)_i \end{bmatrix}$$

and the row vector $(e^h_H)_i$ of length m is given by

$$(e^h_H)_i = \left[\phi^H_1(x_i), \phi^H_2(x_i), \dots \phi^H_m(x_i)\right], \; x_i \in \mathcal{T}^h$$

for $i = 1, \dots, n$. A global coarse-to-fine extension operator I^h_H can be defined as the transpose of I^H_h.

To define the coarse function $T_0 : R^{3n} \to R^{3n}$, we introduce a projection $T^c : R^{3n} \to R^{3m}$ as the solution of the linearize coarse system

$$F^c(x^*_c) + J^c(x^*_c)(T^c(x) - x^*_c) = I^H_h F(x), \tag{2.11}$$

for any given $x \in R^{3n}$. Note that the left hand side of (2.11) is a first order Taylor approximation of $F^c(x)$ at the exact coarse mesh solution, x^*_c. Since $F^c(x^*_c) = 0$, we rewrite (2.11) as

$$T^c(x) = x^*_c + (J^c(x^*_c))^{-1} I^H_h F(x),$$

provided that $J^c(x^*_c)$ is nonsingular. It is easy to see that $T^c(x^*)$ can be computed without knowing the exact solution x^* of F, and $T^c(x^*) = x^*_c$. Then the coarse function can be defined as

$$T_0(x) = I^h_H(T^c(x) - T^c(x^*)) = I^h_H(J^c(x^*_c))^{-1} I^H_h F(x)$$

and its derivative is given by

$$\frac{\partial T_0(x)}{\partial x} = I^h_H(J^c(x^*_c))^{-1} I^H_h J(x). \tag{2.12}$$

We introduce a new nonlinear function

$$\mathcal{F}^{(2)}(x) = T_0(x) + \sum_{i=1}^{N} R^T_i T_i(x),$$

and combining (2.12) and (2.9), we obtain an approximation of Jacobian of $\mathcal{F}^{(2)}$ in the form

$$\widehat{\mathcal{J}}^{(2)}(x) = \left\{ I^h_H(J^c(x^*_c))^{-1} I^H_h + \sum_{i=1}^{N} \left[R^T_i (J_i(x))^{-1} R_i\right] \right\} J(x).$$

The two-level additive Schwarz preconditioned inexact Newton algorithm with a linear coarse solver (ASPIN(2)) is defined as: Find the solution x^* of (2.5) by solving the nonlinearly preconditioned system

$$\mathcal{F}^{(2)}(x) = 0, \tag{2.13}$$

using INB with an initial guess $x^{(0)}$. Details of ASPIN(2) is given below. Let $x^{(0)}$ be an initial guess and $x^{(k)}$ the current approximate solution. Then a new approximate solution $x^{(k+1)}$ can be computed by the ASPIN(2) algorithm as follows:

Step 1: Evaluate the nonlinear residual $\mathcal{F}^{(2)}(x)$ at $x^{(k)}$ through the following steps:

1. Find $w_0^{(k)}$ by solving the linearize coarse mesh problem

$$J^c(x_c^*)z_c = I_h^H F(x^{(k)}) \tag{2.14}$$

using a Krylov-Schwarz method with a left preconditioner, $P^{-1} = \sum_{i=1}^N (R_i^c)^T (J_i^c)^{-1} R_i^c$ and the initial guess $z_c = 0$.
2. Find $w_i^{(k)} = T_i(x^{(k)})$ by solving in parallel, the local nonlinear systems

$$G_i(w) \equiv F_i(x_i^{(k)} - w) = 0 \tag{2.15}$$

using Newton method with backtracking and the initial guess $w = 0$.
3. Form the global residual

$$\mathcal{F}^{(2)}(x^{(k)}) = I_H^h w_0^{(k)} + \sum_{i=1}^N R_i^T w_i^{(k)}.$$

Step 2: Check the stopping condition on $||\mathcal{F}^{(2)}(x^{(k)})||_2$. If $||\mathcal{F}^{(2)}(x^{(k)})||_2$ is small enough, stop, otherwise, continue.

Step 3: Evaluate pieces of the Jacobian matrix $\mathcal{J}^{(2)}(x)$ of the preconditioned system that are needed in order to multiply (2.16) below with a vector in the next step. This includes $J(x^{(k)})$ as well as J_i and its sparse LU factorization.

$$\widehat{\mathcal{J}}^{(2)} = \left\{ I_H^h (J^c(x_c^*))^{-1} I_h^H + \sum_{i=1}^N \left[R_i^T (J_i(x^{(k)}))^{-1} R_i \right] \right\} J(x^{(k)}). \tag{2.16}$$

Step 4: Find an inexact Newton direction $s^{(k)}$ by solving the following Jacobian system approximately using a Krylov subspace method

$$\widehat{\mathcal{J}}^{(2)} s^{(k)} = -\mathcal{F}^{(2)}(x^{(k)}) \tag{2.17}$$

in the sense that

$$||\mathcal{F}^{(2)}(x^{(k)}) + \widehat{\mathcal{J}}^{(2)}(x^{(k)}) s^{(k)}||_2 \leq \eta_k ||\mathcal{F}^{(2)}(x^{(k)})||_2 \tag{2.18}$$

for some $\eta_k \in [0, 1)$.

Step 5: Scale the search direction $s^{(k)} \leftarrow \frac{s_{max}}{||s^{(k)}||_2} s^{(k)}$ if $||s^{(k)}||_2 \geq s_{max}$.

Step 6: Compute a new approximate solution

$$x^{(k+1)} = x^{(k)} + \lambda^{(k)} s^{(k)},$$

where $\lambda^{(k)}$ is determined by the linesearch technique.

Remark 1. No preconditioning is used in Step 4 of ASPIN(2). In fact, $\widehat{\mathcal{J}}^{(2)}$ can be viewed as the original Jacobian system J preconditioned by a two-level additive Schwarz preconditioner, where the coarse preconditioner $I_H^h(J^c(I_H^h x^{(k)} I_h^H)^{-1} I_h^H$ is approximated by $I_H^h(J^c(x_c^*))^{-1} I_h^H$. Hence, $\widehat{\mathcal{J}}^{(2)}$ is well-conditioned through nonlinear preconditioning as long as $I_H^h x^{(k)} I_h^H$ is close to x_c^*.

Remark 2. Although each component of $\widehat{\mathcal{J}}^{(2)}$ is sparse, $\widehat{\mathcal{J}}^{(2)}$ itself is often dense and expensive to form explicitly. However, if a Krylov subspace method is used to the global Jacobian system (2.17), only the Jacobian-vector product, $u = \widehat{\mathcal{J}}^{(2)}v$, is required. In a distributed-memory parallel implementation, this operation consists of five phrases:

1. Solve $J^c(x_c^*)z_c = I_h^H v$, using a Krylov-Schwarz method with a left preconditioner, $P^{-1} = \sum_{i=1}^N (R_i^c)^T (J_i^c)^{-1} R_i^c$ and the initial guess $z_c = 0$.
2. Perform the matrix-vector multiply, $w = Jv$, in parallel.
3. On each subdomain, collect the data from the subdomain and its neighboring subdomains, $w_i = R_i w$.
4. Solve $J_i u_i = w_i$ using a sparse direct solver.
5. Send the partial solutions to its neighboring subdomain and take the sum, $u = \sum_{i=1}^N R_i^T u_i + I_H^h z_c$.

3 Numerical Results

In this section, we consider a two-dimensional lid-driven cavity flow problem. We used PETSc [1] for the parallel implementation and obtained all numerical results on a cluster of workstations. Only machine independent results are reported. In our implementation, after ordering the mesh points, we numbered unknown nodal values in the order of u_1^h, u_2^h, and p^h at each mesh point. The mesh points were grouped subdomain by subdomain for the purpose of parallel processing. Regular checkerboard partitions were used for our experiments. The number of subdomains was always the same as the number of processors, n_p. At the fine mesh level, the linesearch technique [4] was used for both global and local nonlinear problems. The global nonlinear iteration was stopped if the condition $||\mathcal{F}^{(2)}(x^{(k)})||_2 \le 10^{-6}||\mathcal{F}^{(2)}(x^{(0)})||_2$ was satisfied, and the local nonlinear iteration on each subdomain was stopped if the condition $||G_i(w_{i,l}^{(k)})||_2 \le 10^{-4}||G_i(w_{i,0}^{(k)})||_2$ is satisfied. Restarted GMRES(200) was used for solving the global Jacobian systems (2.17). The global linear iteration was stopped if the relative tolerance $||\mathcal{F}^{(2)}(x^{(k)}) + \mathcal{J}^{(2)}(x^{(k)})s^{(k)}||_2 \le 10^{-6}||\mathcal{F}^{(2)}(x^{(k)})||_2$ was satisfied. During local nonlinear iterations, a direct sparse solver, LU decomposition, was employed for solving each local Jacobian system. At the coarse mesh level, restarted GMRES(200) with a left Schwarz preconditioner was used for solving the coarse systems (2.14). The stopping criterion for the coarse mesh problem was that the condition $||I_h^H F(x^{(k)}) - J^c(x_c^*)z_c||_2 \le 10^{-10}||I_h^H F(x^{(k)})||_2$ was satisfied. $\delta = 2$ for both the fine and coarse systems. As suggested in [6], we included the re-scaling of the search direction $s^{(k)}$ in Step 5 if $||s^{(k)}||_2 \ge s_{max}$ to enhance the robustness of ASPIN for solving incompressible flows. This step also reduces the number of line search steps, since the evaluation of nonlinearly preconditioned function is expensive. All numerical results reported here are based on the optimal choice of the parameter s_{max}, which results in the smallest number of global nonlinear iterations.

We first study the effect of the coarse mesh size on the global nonlinear iterations and the global linear iterations of ASPIN(2) for different values of Reynolds number. In this set of numerical experiments, all results are obtained using a fixed fine mesh 128×128 on 16 processors, and the coarse mesh size is varied from 16×16 to 80×80.

Table 1. ASPIN(2): Varying the coarse mesh size for different values of Reynolds number. Fine mesh: 128×128. The number of processors n_p=16

Coarse meshes	Re=10^3	Re=3×10^3	Re=5×10^3	Re=8×10^3	Re=10^4
	Number of global nonlinear iterations				
16×16	8	11	11	14	17
20×20	9	9	11	13	14
32×32	8	9	11	10	12
40×40	8	9	9	10	11
64×64	8	10	9	11	11
	Average number of global linear iterations				
16×16	58	74	94	111	122
20×20	50	66	75	89	103
32×32	45	52	59	64	68
40×40	43	50	54	60	60
64×64	42	49	52	55	65

Table 2. ASPIN(1) and ASPIN(2): Varying the number of processors. Fine mesh size: 128×128. Coarse mesh size: 40×40

n_p	Re=10^3	Re=3×10^3	Re=5×10^3	Re=8×10^3	Re=10^4
	Number of global nonlinear iterations				
ASPIN(1)					
$2 \times 2 = 4$	9	10	13	19	19
$4 \times 4 = 16$	9	12	12	16	18
$8 \times 8 = 64$	10	15	14	19	19
ASPIN(2)					
$2 \times 2 = 4$	9	9	11	10	12
$4 \times 4 = 16$	8	9	9	10	11
$8 \times 8 = 64$	8	9	12	12	14
	Average number of global linear iterations				
ASPIN(1)					
$2 \times 2 = 4$	67	69	71	73	74
$4 \times 4 = 16$	127	128	133	137	140
$8 \times 8 = 64$	395	394	400	497	655
ASPIN(2)					
$2 \times 2 = 4$	33	40	40	40	46
$4 \times 4 = 16$	43	50	54	60	60
$8 \times 8 = 64$	49	62	61	78	79

Table 1 shows that to apply two-level methods on a moderate number of processors, the coarse mesh has to be sufficiently fine, say 40×40 in this case. For this particular case, the numbers of global nonlinear iterations, as well as global linear iterations, are not very sensitive with the increase of Reynolds number. To study the parallel scalability of ASPIN(2) with respect to the number of processors, we use a fixed fine mesh 128×128 and a coarse mesh 40×40. For comparison purposes, we also include the results obtained using ASPIN(1). Table 2 shows that by adding a coarse preconditioner, not only the global linear iterations is reduced significantly as we increase the number of processors from 4 to 64, but also the global nonlinear iterations is improved especially for high Reynolds number flows.

References

1. S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *Portable, Extensible, Toolkit for Scientific Computation(PETSc) home page*, http://www.mcs.anl.gov/petsc, 2004.
2. X.-C. Cai and D. E. Keyes, *Nonlinearly preconditioned inexact Newton algorithms*, SIAM J. Sci. Comput., 24 (2002), pp. 183-200.
3. X.-C. Cai, D. E. Keyes, and L. Marcinkowski, *Nonlinear additive Schwarz preconditioners and applications in computational fluid dynamics*, Int. J. Numer. Meth. Fluids, 40 (2002), pp. 1463-1470.
4. J. Dennis and R. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, Philadelphia, 1996.
5. L. P. Franca and S. L. Frey, *Stabilized finite element method: II. The incompressible Navier-Stokes equation*, Comput. Methods Appl. Mech. Engrg., 99 (1992), pp. 209-233.
6. F.-N. Hwang and X.-C. Cai, *A parallel nonlinear additive Schwarz preconditioned inexact Newton algorithm for incompressible Navier-Stokes equations*, J. Comput. Phys., (2004), to appear.
7. L. Marcinkowski and X.-C. Cai, *Parallel performance of some two-level ASPIN algorithms*, Lecture Notes in Computational Science and Engineering, ed. R. Kornhuber, R. H. W. Hoppe, D. E. Keyes, J. Periaux, O. Pironneau and J. Xu, Springer-Verlag, Haidelberg, pp. 639-646.
8. J. N. Shadid, R. S. Tuminaro, and H. F. Walker, *An inexact Newton method for fully coupled solution of the Navier-Stokes equations with heat and mass transport*, J. Comput. Phys., 137 (1997), pp. 155-185.

Structure-Preserving Model Reduction

Ren-Cang Li[1] and Zhaojun Bai[2]

[1] Department of Mathematics, University of Kentucky, Lexington, KY 40506, USA
rcli@ms.uky.edu
[2] Department of Computer Science and Department of Mathematics
University of California, Davis, CA 95616, USA
bai@cs.ucdavis.edu

Abstract. A general framework for structure-preserving model reduction by Krylov subspace projection methods is developed. The goal is to preserve any substructures of importance in the matrices L, G, C, B that define the model prescribed by transfer function $H(s) = L^*(G + sC)^{-1}B$. Many existing structure-preserving model-order reduction methods for linear and second-order dynamical systems can be derived under this general framework.

1 Introduction

Krylov subspace projection methods are increasingly popular in model reduction owing to their numerical efficiency for very large systems, such as those arising from structural dynamics, control systems, circuit simulations, computational electromagnetics and microelectromechanical systems. Recent survey articles [1,2,7] provide in depth review of the subject and comprehensive references. Roughly speaking, these methods project the original system onto a smaller subspace to arrive at a (much) smaller system having properties, among others, that many leading terms (called *moments*) of the associated (matrix-valued) transfer functions expanded at given points for the original and reduced systems match.

Consider the matrix-valued transfer function of the form

$$H(s) = L^*(G + sC)^{-1}B, \tag{1.1}$$

which describes an associated multi-input multi-output (MIMO) time-invariant system to be studied. Here $G, C \in \mathbb{C}^{N\times N}$, $B \in \mathbb{C}^{N\times m}$, $L \in \mathbb{C}^{N\times p}$. In today's applications of interests, such as VLSI circuit designs and structural dynamics, N can be up to millions [1,2,7]. Computations of $H(s)$ usually have to be done through some kind of reduction on L, G, C and B. Let $X, Y \in \mathbb{C}^{N\times n}$ such that Y^*GX is nonsingular (and thus $\text{rank}(X) = \text{rank}(Y) = n$). We may reduce the transfer function $H(s)$ to

$$H_{\text{R}}(s) = L_{\text{R}}^*(G_{\text{R}} + sC_{\text{R}})^{-1}B_{\text{R}}, \tag{1.2}$$

where

$$L_{\text{R}} = X^*L,\ G_{\text{R}} = Y^*GX,\ C_{\text{R}} = Y^*CX,\ B_{\text{R}} = Y^*B. \tag{1.3}$$

There are various techniques to pick X and Y to perform reductions. Among them Krylov subspace-based model-reduction is getting much of the attention. The idea is to pick X and Y as the bases of properly defined Krylov subspaces so that $H(s) = H_{\rm R}(s) + \mathcal{O}(s^{\ell+1})$. In this paper, we show, in addition, X and Y can be chosen specially enough to preserve meaningful physical substructures.

2 Framework for Structure-Preserving Model Reduction

Suppose the matrices L, G, C, B in the transfer function (1.1) have some natural partitioning that is derived from, e.g., the physical layout of a VLSI circuit or a structural dynamical system:

$$G = \begin{array}{c c} & \begin{array}{cc} N_1 & N_2 \end{array} \\ \begin{array}{c} N_1' \\ N_2' \end{array} & \begin{pmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{pmatrix} \end{array}, \quad C = \begin{array}{c c} & \begin{array}{cc} N_1 & N_2 \end{array} \\ \begin{array}{c} N_1' \\ N_2' \end{array} & \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \end{array}, \tag{2.1}$$

$$L = \begin{array}{c c} & p \\ \begin{array}{c} N_1 \\ N_2 \end{array} & \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} \end{array}, \quad B = \begin{array}{c c} & m \\ \begin{array}{c} N_1' \\ N_2' \end{array} & \begin{pmatrix} B_1 \\ B_2 \end{pmatrix} \end{array}, \tag{2.2}$$

where $N_1' + N_2' = N_1 + N_2 = N$. We wish that the reduced system would inherit the same structure; that is, $L_{\rm R}$, $G_{\rm R}$, $C_{\rm R}$ and $B_{\rm R}$ could be partitioned so that

$$G_{\rm R} = \begin{array}{c c} & \begin{array}{cc} n_1 & n_2 \end{array} \\ \begin{array}{c} n_1' \\ n_2' \end{array} & \begin{pmatrix} G_{\rm R11} & G_{\rm R12} \\ G_{\rm R21} & G_{\rm R22} \end{pmatrix} \end{array}, \; C_{\rm R} = \begin{array}{c c} & \begin{array}{cc} n_1 & n_2 \end{array} \\ \begin{array}{c} n_1' \\ n_2' \end{array} & \begin{pmatrix} C_{\rm R11} & C_{\rm R12} \\ C_{\rm R21} & C_{\rm R22} \end{pmatrix} \end{array}, \tag{2.3}$$

$$L_{\rm R} = \begin{array}{c c} & p \\ \begin{array}{c} n_1 \\ n_2 \end{array} & \begin{pmatrix} L_{\rm R1} \\ L_{\rm R2} \end{pmatrix} \end{array}, \; B_{\rm R} = \begin{array}{c c} & m \\ \begin{array}{c} n_1' \\ n_2' \end{array} & \begin{pmatrix} B_{\rm R1} \\ B_{\rm R2} \end{pmatrix} \end{array}, \tag{2.4}$$

with each sub-block a direct reduction from the corresponding sub-block in the original system, e.g., $G_{\rm R11}$ from G_{11}, where $n_1 + n_2 = n_1' + n_2'$. In the formulation (1.3) for the reduced system, this can be accomplished by picking

$$X = \begin{array}{c c} & \begin{array}{cc} n_1 & n_2 \end{array} \\ \begin{array}{c} N_1 \\ N_2 \end{array} & \begin{pmatrix} X_1 & 0 \\ 0 & X_2 \end{pmatrix} \end{array}, \quad Y = \begin{array}{c c} & \begin{array}{cc} n_1' & n_2' \end{array} \\ \begin{array}{c} N_1' \\ N_2' \end{array} & \begin{pmatrix} Y_1 & 0 \\ 0 & Y_2 \end{pmatrix} \end{array}, \tag{2.5}$$

such that $\operatorname{rank}(X_j) = n_j$, $\operatorname{rank}(Y_i) = n_i'$. Then the submatrices of the coefficient matrices $L_{\rm R}$, $G_{\rm R}$, $C_{\rm R}$ and $B_{\rm R}$ of the reduced system are given by

$$L_{{\rm R}j} = X_j^* L_j, \; G_{{\rm R}ij} = Y_i^* G_{ij} X_j, \; C_{{\rm R}ij} = Y_i^* C_{ij} X_j, \; B_{{\rm R}i} = Y_i^* B_i. \tag{2.6}$$

A reduction as in (2.3) – (2.6) is conceivably useful for the system matrices with meaningful substructures. For example, for the time-domain modified nodal analysis (MNA) circuit equations targeted by PRIMA [12] and SyMPVL [6], system matrices have the following natural partitioning (adopting the formulation in [6])

$$G = \begin{pmatrix} G_{11} & G_{12} \\ G_{12}^* & 0 \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & 0 \\ 0 & -C_{22} \end{pmatrix}, \quad G_{11}^* = G_{11}, \quad C_{ii}^* = C_{ii}, \quad L = B, \tag{2.7}$$

where all sub-matrices have their own physical interpretations. As proved in [12], reduction (1.3) (and thus (2.6) included) with $Y = X$ preserves passivity of the system (2.7).

Remark 1. This substructural preserving model reduction technique (2.3) – (2.6) was inspired by Su and Craig [17] concerning a second-order system which can always be equivalently turned into a linear system (see (4.2) in the next section) with a natural partitioning just as in (2.1) and (2.2).

Define the *kth Krylov subspace* generated by $A \in \mathbb{C}^{N\times N}$ on $Z \in \mathbb{C}^{N\times \ell}$ as

$$\mathcal{K}_k(A, Z) \overset{\text{def}}{=} \mathsf{span}(Z, AZ, \dots, A^{k-1}Z),$$

where $\mathsf{span}(\cdots)$ is the subspace spanned by the columns of argument matrices.

Theorem 1. *Assume that G and G_{R} are nonsingular (and thus the total number of columns in all X_i and that in all Y_i must be the same).*

1. *If $\mathcal{K}_k(G^{-1}C, G^{-1}B) \subseteq \mathsf{span}(X)$ and $Y = X$, then $H(s) = H_{\mathrm{R}}(s) + \mathcal{O}(s^k)$.*
2. *If G and C are Hermitian, and if $\mathcal{K}_k\left(G^{-1}C, G^{-1}(B\ \ L)\right) \subseteq \mathsf{span}(X)$ and $Y = X$, then $H(s) = H_{\mathrm{R}}(s) + \mathcal{O}(s^{2k})$.*
3. *If $\mathcal{K}_k(G^{-1}C, G^{-1}B) \subseteq \mathsf{span}(X)$ and $\mathcal{K}_r(G^{-*}C^*, G^{-*}L) \subseteq \mathsf{span}(Y)$, then $H(s) = H_{\mathrm{R}}(s) + \mathcal{O}(s^{k+r})$.*

Remark 2. Theorem 1 in its generality is due to [8]. It is an extension of similar theorems in [19] for $C = I$ and $G^{-1}B = b$ (vector). For a simpler and cleaner proof based on the projector language, the reader is referred to [10].

Now that we have shown the substructures in (2.1) and (2.2) can be preserved via (2.3) – (2.6). But this is for approximating $H(s)$ around $s = 0$ only and in practice approximations to $H(s)$ around other selected points $s_0 \neq 0$ may be sought, too. Can a shift be incorporated without destroying the existing substructures? The answer is *yes*. Let s_0 be a shift and write

$$s = s_0 + (s - s_0), \tag{2.8}$$

and then

$$G + sC = G + s_0C + (s - s_0)C \overset{\text{def}}{=} G(s_0) + \widetilde{s}C. \tag{2.9}$$

Upon substitutions (i.e., renaming)

$$G(s_0) \to G, \quad \widetilde{s} \to s,$$

the problem of approximating $H(s)$ around $s = s_0$ becomes equivalently to approximate the substituted $H(s)$ around $s = 0$. Observe that any reduction on $G(s_0)$ and C by $Y^*G(s_0)X$ and Y^*CX can be done through reducing G and C directly as in (1.3) because

$$G_{\mathrm{R}}(s_0) \overset{\text{def}}{=} Y^*G(s_0)X = Y^*GX + s_0Y^*CX = G_{\mathrm{R}} + s_0C_{\mathrm{R}}. \tag{2.10}$$

This is a significant observation because it says that even for approximating $H(s)$ near a different point $s_0 \neq 0$, reduction can still be done directly to the original matrices L, G, C, and B, regardless of the shift (2.8).

As a straightforward application of Theorem 1, we have the following theorem.

Theorem 2. *Let integers $k, r \geq 0$, and let $G(s_0)$ be defined as in* (2.9). *Assume that $G(s_0)$ and $G_{\rm R}(s_0)$ are nonsingular.*

1. *If $\mathcal{K}_k(G(s_0)^{-1}C, G(s_0)^{-1}B) \subseteq \mathsf{span}(X)$ and $Y = X$, then $H(s) = H_{\rm R}(s) + \mathcal{O}((s-s_0)^k)$.*
2. *For real s_0, if G and C are Hermitian, and if $\mathcal{K}_k\left(G^{-1}(s_0)C, G(s_0)^{-1}(B\ \ L)\right) \subseteq \mathsf{span}(X)$ and $Y = X$, then $H(s) = H_{\rm R}(s) + \mathcal{O}((s-s_0)^{2k})$.*
3. *If $\mathcal{K}_k(G(s_0)^{-1}C, G(s_0)^{-1}B) \subseteq \mathsf{span}(X)$ and $\mathcal{K}_r(G(s_0)^{-*}C^*, G(s_0)^{-*}L) \subseteq \mathsf{span}(Y)$, then $H(s) = H_{\rm R}(s) + \mathcal{O}((s-s_0)^{k+r})$.*

A sample Arnoldi-type implementation to realize Item 1 of this theorem is given below, where **strAMR** stands for *structural preserving Arnoldi-type model reduction*. Sample implementations to Items 2 and 3 can be given in a similar way.

---

strAMR – Sample Implementation:
Given L, G, C, B as in (2.1) and (2.2), and expansion point s_0.

```
1. $\widehat{G} = G + s_0 C$; solve $\widehat{G}\widehat{Q} = B$ for $\widehat{Q}$;
2. $Q_1 = \mathtt{orth}(\widehat{Q})$: an orthonormal basis matrix for $\mathsf{span}(\widehat{G}^{-1}B)$;
3. Arnoldi process computes $\widetilde{X}$:

   For $j = 1$ to $k-1$ do
       Solve $\widehat{G}\widehat{Q} = CQ_j$ for $\widehat{Q}$;
       For $i = 1$ to $j$ do
           $\widehat{Q} = \widehat{Q} - Q_i(Q_i^*\widehat{Q})$;
       EndFor
       $Q_{j+1} = \mathtt{orth}(\widehat{Q})$;
   EndFor

   Partition $\widetilde{X} = (Q_1\ Q_2\ \cdots\ Q_k)$ as $\widetilde{X} = \begin{array}{c} {\scriptstyle N_1} \\ {\scriptstyle N_2} \end{array}\begin{pmatrix} \widetilde{X}_1 \\ \widetilde{X}_2 \end{pmatrix}$;

4. $X_1 = \mathtt{orth}(\widetilde{X}_1)$; $X_2 = \mathtt{orth}(\widetilde{X}_2)$; $Y_i = X_i$;
5. Compute nonzero blocks of $L_{\rm R}$, $G_{\rm R}$, $C_{\rm R}$, and $B_{\rm R}$, as in (2.6);
6. Evaluate the reduced $H_{\rm R}(s)$ as needed.
```

Remark 3. The invariance property (2.10) of the reduction on L, G, C, and B regardless of the shift (2.8) makes it possible to match moments at multiple points by one reduction. This is done by enforcing $\mathsf{span}(X)$ and/or $\mathsf{span}(Y)$ containing more appropriate Krylov subspaces associated at multiple points. To avoid repetition, we shall omit explicitly stating it. See [8] and Ruhe [13,14].

3 Structures of Krylov Subspaces of Block Matrices

The results of this section are of general interest. The matrices here do not necessarily have anything to do with the transfer function. Consider

$$A = \begin{array}{c|cc} & N_1 & N_2 \\ \hline N_1 & A_{11} & A_{12} \\ N_2 & A_{21} & A_{22} \end{array}, \quad B = \begin{array}{c|c} & m \\ \hline N_1 & B_1 \\ N_2 & B_2 \end{array}, \tag{3.1}$$

where $N_1 + N_2 = N$. The following theorem describes the structures in a basis matrix of $\mathcal{K}_k(A, B)$ when one of A_{ij}'s is zero.

Theorem 3. *Let A and B be partitioned as in* (3.1)*, and let* $\mathsf{span}(\widetilde{X}) = \mathcal{K}_k(A, B)$ *be partitioned as*

$$\widetilde{X} = \begin{array}{c|cc} & n_1' & n_2' \\ \hline N_1 & \widetilde{X}_{11} & \widetilde{X}_{12} \\ N_2 & \widetilde{X}_{21} & \widetilde{X}_{22} \end{array} \equiv \begin{array}{c|c} & n_1'+n_2' \\ \hline N_1 & \widetilde{X}_1 \\ N_2 & \widetilde{X}_2 \end{array}$$

such that $\mathsf{span}\begin{pmatrix} \widetilde{X}_{11} \\ \widetilde{X}_{21} \end{pmatrix} = \mathcal{K}_{k-1}(A, B)$*, and let $\alpha \neq 0$ be a scalar which may be different at different occurrences. Then*

1. *If* $A_{11} = 0$*, then* $\mathsf{span}(\widetilde{X}_1) = \mathsf{span}(B_1 \;\; A_{12}\widetilde{X}_{21}) \subseteq \mathsf{span}(B_1 \;\; A_{12}\widetilde{X}_2)$*. If in addition* $A_{12} = \alpha I$ *(and thus* $N_1 = N_2$*),* $\mathsf{span}(\widetilde{X}_1) = \mathsf{span}(B_1 \;\; \widetilde{X}_{21}) \subseteq \mathsf{span}(B_1 \;\; \widetilde{X}_2)$*.*
2. *If* $A_{12} = 0$*, then* $\mathsf{span}(\widetilde{X}_1) = \mathcal{K}_k(A_{11}, B_1)$*.*
3. *If* $A_{21} = 0$*, then* $\mathsf{span}(\widetilde{X}_2) = \mathcal{K}_k(A_{22}, B_2)$*.*
4. *If* $A_{22} = 0$*, then* $\mathsf{span}(\widetilde{X}_2) = \mathsf{span}(B_2 \;\; A_{21}\widetilde{X}_{11}) \subseteq \mathsf{span}(B_2 \;\; A_{21}\widetilde{X}_1)$*. If in addition* $A_{21} = \alpha I$ *(and thus* $N_1 = N_2$*),* $\mathsf{span}(\widetilde{X}_2) = \mathsf{span}(B_2 \;\; \widetilde{X}_{11}) \subseteq \mathsf{span}(B_2 \;\; \widetilde{X}_1)$*.*

Proof: All claims are consequences of the following observation:

$$\text{if} \quad A^i B = \begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix}, \quad \text{then} \quad A^{i+1} B = \begin{pmatrix} A_{11}Z_1 + A_{12}Z_2 \\ A_{21}Z_1 + A_{22}Z_2 \end{pmatrix}.$$

Then combining the assumption that one of $A_{ij} = 0$ will complete the proof. □

Item 4 of Theorem 3 was implicitly stated in [3,4,17]. It gives a relation between $\mathsf{span}(\widetilde{X}_1)$ and $\mathsf{span}(\widetilde{X}_2)$; so does Item 1. It is Item 4 that led to structure-preserving dimension reductions of second-order systems. See § 4.

4 Model Reduction of Second-Order Systems

In this section, we show how to apply the theory presented in the previous sections to the structure-preserving model reduction of a second-order system. Consider the transfer function of a second-order system

$$H(s) = (V^* + sT^*)(s^2 M + sD + K)^{-1} R, \tag{4.1}$$

where $M, D, K \in \mathbb{C}^{N\times N}$, $R \in \mathbb{C}^{N\times m}$, $T, V \in \mathbb{C}^{N\times p}$. Notation here is adopted from structural dynamics, where M, D, K are mass, damping, and stiffness matrices and are usually Hermitian, but can be non-Hermitian at times. It is quite common to deal with (4.1) by a linearization technique to turn it into the form of (1.1). This is done by setting

$$C = \begin{pmatrix} D & M \\ M & 0 \end{pmatrix}, \; G = \begin{pmatrix} K & 0 \\ 0 & -M \end{pmatrix}, \; L = \begin{pmatrix} V \\ T \end{pmatrix}, \; B = \begin{pmatrix} R \\ 0 \end{pmatrix}. \tag{4.2}$$

By now, all existing developments for the transfer function (1.1) can be applied in a straightforward way, but then reduced models likely lose the second-order characteristics, i.e., they may not be turned into the second-order transfer functions[3] and consequently the reduced models have little physical significance. To overcome this, Su and Craig [17] made an important observation about the structures of the associated Krylov subspaces that make it possible for the reduced second-order system to still have the second-order form

$$H_R(s) = (V_R^* + sT_R^*)(s^2 M_R + sD_R + K_R)^{-1} R_R, \tag{4.3}$$

where

$$\begin{array}{l} M_R = Y_1^* M X_1, \; D_R = Y_1^* D X_1, \; K_R = Y_1^* K X_1, \\ V_R = X_1^* V, \; T_R = X_1^* T, \; R_R = Y_1^* R. \end{array} \tag{4.4}$$

and $X_1, Y_1 \in \mathbb{C}^{N\times n}$ having full column rank. Together with L, G, C and B as defined by (4.2), the transfer functions $H(s)$ and $H_R(s)$ of (4.1) and (4.3) takes the forms (1.1) and (1.2) with (1.3), and

$$X = \begin{array}{c|cc} & n & n \\ N & X_1 & 0 \\ N & 0 & X_1 \end{array} \quad \text{and} \quad Y = \begin{array}{c|cc} & n & n \\ N & Y_1 & 0 \\ N & 0 & Y_1 \end{array}. \tag{4.5}$$

Reduction as such for the second-order system falls nicely into our framework in §2. The question is how to construct X and Y, noticing the differences in X and Y between (2.5) and (4.5). This is where Theorem 3 comes in to help. A sample Arnoldi-type implementation **qAMR** is as follows. For more detail, see [10]. Another implementation includes the original one of [17].

5 Numerical Examples

The first example is taken from [16]. Here $N = 256$, The structure of G and C are as in Figure 1, $N_i' = N_i = 128$ ($i = 1, 2$), $p = m = 1$, and L and B are randomly chosen.

We compare the approximate accuracy of the "*structurally reduced*" models by **strAMR** as proposed against otherwise "*generally reduced*" ones, i.e., **strAMR** without Step 4 (and therefore $X = \widetilde{X}$). Figure 2 plots the values of the original and reduced transfer functions and the relative errors of the reduced functions, where $Y = X$ and $\mathsf{span}(X) \supset \mathcal{K}_{20}(G^{-1}C, G^{-1}B)$. It clearly shows that the structurally reduced model is more accurate in the long range of frequency.

[3] It is possible to turn a linear system of even dimension into a second-order system. Recently [11,15] and [9] propose two different ways to do that; but in both cases the coefficient matrices of the resulted second-order system cannot be related to the original system in a meaningful way.

qAMR – Sample Implementation: Computing X_1.

1. Compute $\widehat{X}$ such that $\mathcal{K}_q(G^{-1}C, G^{-1}(B \;\; L) \subseteq \mathsf{span}(\widehat{X})$, by, e.g., **strAMR**;
2. Partition $\widehat{X} = \begin{array}{c} N \\ N \end{array} \begin{pmatrix} \widehat{X}_1 \\ \widehat{X}_2 \end{pmatrix}$;
3. Compute $X_1 = \texttt{orth}((\widehat{X}_1 \quad M^{-1}T))$.

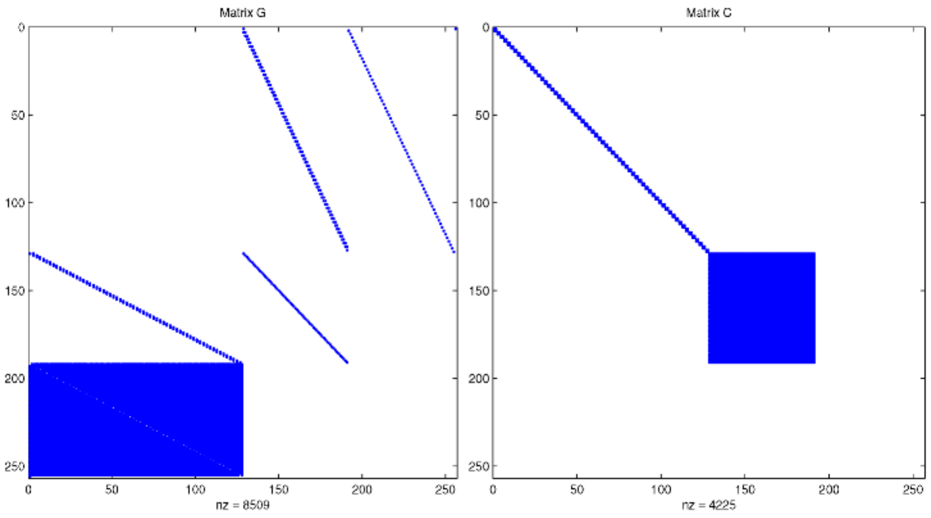


Fig. 1. Block Structure of G (left) and C (right)

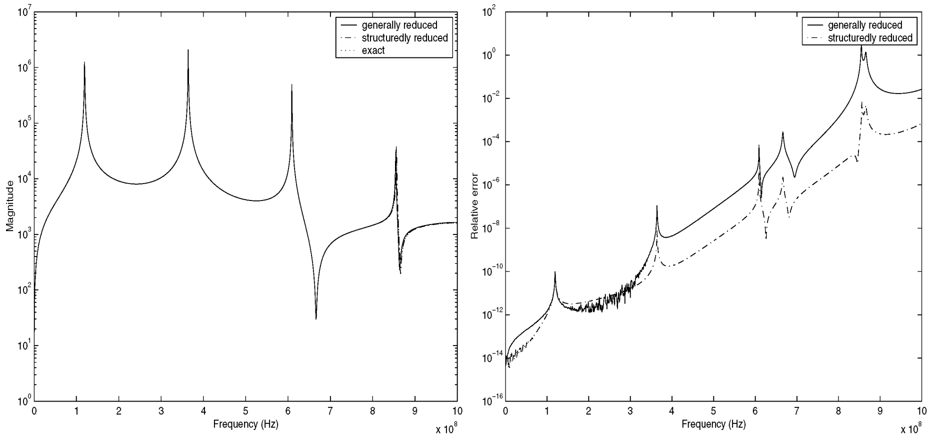


Fig. 2. Transfer functions (left) and relative errors (right)

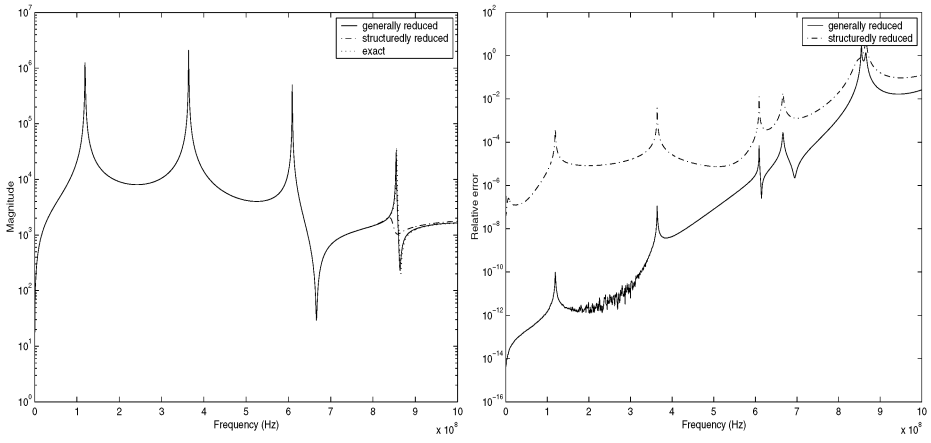


Fig. 3. Transfer functions (left) and relative errors (right); Preserving incorrect structure can lead less accurate approximations

It is natural to wonder whether incorrect structural partitioning would make any difference. Indeed it does. Supposedly we take $N_1' = N_1 = 128 + 20$ and $N_2' = N_2 = 128 - 20$. Figure 3 plots the same things as Figure 2, except with this new partition, where again $Y = X$ and $\mathsf{span}(X) \supset \mathcal{K}_{20}(G^{-1}C, G^{-1}B)$. This figure shows that improper partitioning can degrade accuracy. But more than that, for this partitioning "*structural reduction*" is less accurate than the "*general reduction*" which is quite counter-intuitive and surprising because $\mathsf{span}(X)$ with some partitioning includes $\mathsf{span}(X)$ without any partitioning, and thus a reduction with partitioning should do at least just as well as one without in terms of accuracy – further studies needed.

Next example is the second-order system from [2, §3.3]: $N = 400$, $p = m = 1$, $T = 0$, and $V = R$ randomly chosen. Figure 4 plots the values of the original and reduced transfer functions and relative errors, where "*quadratically reduced*" refers to (4.3) with (4.4) and X_1 by, e.g., **qAMR**, and "*linearly reduced*" refers to (1.2) and (1.3) through linearization (4.2) with $Y = X(= \widehat{X}$ in **qAMR** without Step 3).

6 Conclusions

A general framework for structural model reduction is established. Existing technique of Su and Craig for the second-order system can be easily realized within the framework. The idea is extensible to block partitioning with more than 2-by-2 blocks and thus makes it possible to conserve sub-structures as fine as needed for any particular system. The idea about the structures of Krylov subspaces of block matrices is not limited to 2-by-2 blocks as in Theorem 3, either and consequently the development outlined in §4 is extensible to systems of order higher than 2. Detail is in [10]. Numerical examples show the worth of the idea, as well as that incorrect identification of structures can result in poor numerical accuracy.

The work of Su and Craig [17] has spawned several recent research papers on model reduction of second-order systems and quadratic eigenvalue problems, includ-

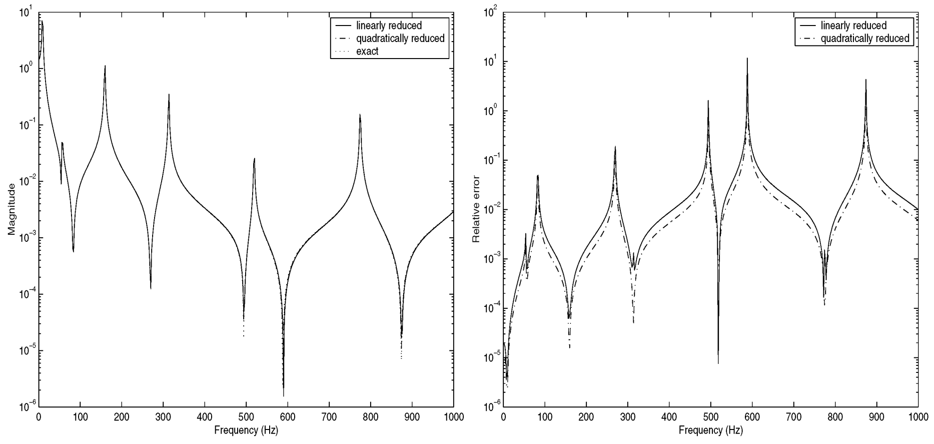


Fig. 4. Transfer functions (left) and relative errors (right): a second-order example

ing [3,4,5,18]. But the attempt to preserve meaningful substructures as in (2.3) – (2.6) for any general linear systems, not necessarily from linearizing a second-order system, appears to be conceived first by [10].

Acknowledgments

Li and Bai are supported in part by NSF CAREER award under grant CCR-9875201 and NSF ITR grant ACI-0220104, respectively.

References

1. A. C. Antoulas, D. C. Sorensen, and S. Gugercin. A survey of model reduction methods for large-scale systems. In Vadim Olshevsky, editor, *Structured Matrices in Mathematics, Computer Science, and Engineering I: Proceedings of an AMS-IMS-SIAM joint summer research conference, University of Co0lorado, Boulder, June 27–July 1, 1999*, volume 280 of *Comtemporary Mathematics*, pages 193–219. American Mathematical Society, Providence, Rhode Island, 2001.
2. Zhaojun Bai. Krylov subspace techniques for reduced-order modeling of large-scale dynamical systems. *Applied Numerical Mathematics*, 43:9–44, 2002.
3. Zhaojun Bai and Yangfeng Su. SOAR: A second-order Arnoldi method for the solution of the quadratic eigenvalue problem. Computer Science Technical Report CSE-2003-21, University of California, Davis, California, *SIAM J. Matrix Anal. Appl.*, 26(3):640–659, 2003.
4. Zhaojun Bai and Yangfeng Su. Dimension reduction of second-order dynamical systems via a second-order Arnoldi method. Computer Science Technical Report CSE-2004-1, University of California, Davis, California, *SIAM J. Sci. Comp.*, 26(5):1692–1709, 2004.
5. R. Freund. Pade-type reduced-order modeling of higher-order systems. Presentation at Oberwolfach Mini-Workshop on Dimensional Reduction of Large-Scale Systems, October, 2003.

6. R. W. Freund and P. Feldmann. The SyMPVL algorithm and its applications to interconnect simulation. In *Proc. 1997 International Conference on Simulation of Semiconductor Processes and Devices*, pages 113–116, Piscataway, New Jersey, 1997. IEEE.
7. Roland W. Freund. Model reduction methods based on Krylov subspaces. *Acta Numerica*, 12:267–319, 2003.
8. E. J. Grimme. *Krylov Projection Methods For Model Reduction*. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1997.
9. Leonard Hoffnung. *Subspace Projection Methods for the Quadratic Eigenvalue Problem*. PhD thesis, University of Kentucky, Lexington, KY, August 2004.
10. Ren-Cang Li and Zhaojun Bai. Structure-preserving model reductions using a Krylov subspace projection formulation. Technical Report CSE-2004-24, Department of Computer Science, University of California, Davis, *Comm. Math. Sci.*, 3(2):179–199, 2004.
11. D.G. Meyer and S. Srinivasan. Balancing and model reduction for second-order form linear systems. *IEEE Transactions on Automatic Control*, 41(11):1632–1644, November 1996.
12. A. Odabasioglu, M. Celik, and L. T. Pileggi. PRIMA: passive reduced-order interconnect macromodeling algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(8):645–654, August 1998.
13. Axel Ruhe. Rational Krylov sequence methods for eigenvalue computation. *Linear Algebra and Its Applications*, 58:391–405, 1984.
14. Axel Ruhe. Rational Krylov algorithms for nonsymmetric eigenvalue problems. ii. matrix pairs. *Linear Algebra and Its Applications*, 197-198:283–295, 1994.
15. B. Salimbahrami and B. Lohmann. Structure preserving order reduction of large scale second order systems. In *Proceeding of 10th IFAC/IFORS/IMACS/IFIP Symposium on Large Scale Systems: Theory and Applications*, pages 245–250, Osaka, Japan, July 2004.
16. Rodney Daryl Slone. A computationally efficient method for solving electromagnetic interconnect problems: the Padé approximation via the Lanczos process with an error bound. Master's thesis, University of Kentucky, Lexington, KY, 1997.
17. T.-J. Su and R. R. Craig. Model reduction and control of flexible structures using Krylov vectors. *J. Guidance, Control, and Dynamics*, 14(2):260–267, 1991.
18. A. Vandendorpe and P. Van Dooren. Krylov techniques for model reduction of second-order systems. Unpublished Note, March 2, 2004.
19. C. E. Villemagne and R. E. Skelton. Model reduction using a projection formulation. *Int. J. Control*, 46(6):2141–2169, 1987.

A Comparison of Parallel Preconditioners for the Sparse Generalized Eigenvalue Problems by Rayleigh-Quotient Minimization

Sangback Ma[1] and Ho-Jong Jang[2]

[1] School of Electrical Engineering and Computer Science
Hanyang University, Ansan, Korea
sangback2001@empal.com
[2] Department of Mathematics
Hanyang University, Seoul, Korea
hjang@hanyang.ac.kr

Abstract. In this paper we address ourselves to the problem of finding efficient parallel preconditioner for the interior generalized eigenvalue problem $Ax = \lambda Bx$, where A and B are large sparse symmetric positive definite matrices. We consider incomplete LU(ILU)(0) in two variants, Multi-Color block successive over-relaxation(SOR), and Point-symmetric SOR(SSOR). Our results show that for small number of processors the Multi-Color ILU(0) gives the best performance, while for large number of processors the Multi-Color Block SOR does.

1 Introduction

Recently, there has been much efforts to find the interior eigenvalues of the generalized eigenproblem by iterative algorithms based on the optimization of the Rayleigh quotient, and the conjugate gradient(CG) method for the optimization of the Rayleigh quotient has been proven a very attractive and promising technique for large sparse eigenproblems [1,3,4,11,12]. Such applications arise in many cases, e.g., structural mechanics, computational chemistry and plasma physics.

As in the case of a system of linear equations, successful application of the CG method to eigenproblem depends upon the preconditioning techniques. We also need a suitable preconditioner for the parallel processing. ILU(0) is a very popular technique among the various preconditioning techniques, but it has limitations in the parallelization, since it is inherently *serial*. We consider ILU(0) with Muti-Coloring and wavefront techniques to increase parallelism of preconditioner for eigenproblem.

In the present paper we calculate the leftmost m eigenpairs ($m = 5$) and for finite difference matrices with much larger dimensions N ($128^2 \leq N \leq 512^2$). We follow the approach developed in [4] which is based on the minimization of the Rayleigh quotient over a subspace of orthogonal vectors, obtained with a conveniently preconditioned CG(PCG) method. This approach is parallelized and used for the evalution of m leftmost eigenpairs of test matrices coming from finite difference discretizations(FDM).

All computing was performed on a Cray-T3E in Electronics and Telecommunications Research Institute(ETRI), Korea. Cray-T3E is a massively parallel message-passing machine with the 136 individual processing node(PE)s interconnected in a 3D-Torus structure. Each PE, a DEC Alpha EV5.6 chip, is capable of delivering up to 900 Megaflops, amounting to 115 GigaFlops in total. Each PE has 128 MBs core memory.

2 Generalized Eigenproblem via Preconditioned CG

2.1 Conjugate Gradient Method

We shall be concerned with computing a few of the smallest eigenvalues and their corresponding eigenvectors of the generalized eigenvalue problem

$$Ax = \lambda Bx, \tag{2.1}$$

where A and B are large sparse symmetric positive definite matrices of order N. We look for the m smallest eigenvalues

$$0 < \lambda_1 < \lambda_2 \leq \lambda_3 \leq \cdots \leq \lambda_m$$

and for the corresponding eigenvectors $z_1, z_2, \cdots, z_m$ of (1) such that

$$Az_i = \lambda_i B z_i, \;\; {z_i}^T B z_i = 1, \;\; i = 1, 2, \cdots, m. \tag{2.2}$$

The number m of the desired eigenpairs (λ_i, z_i) is much smaller than the order N of the matrices.

We recall that the eigenvectors of (1) are the stationary points of the Rayleigh quotient

$$R(x) = \frac{x^T A x}{x^T B x}, \tag{2.3}$$

and the gradient of $R(x)$ is given by

$$g(x) = \frac{2}{x^T B x}[Ax - R(x)B\,x].$$

For an iterate $x^{(k)}$, the gradient of $R(x^{(k)})$,

$$\nabla R(x^{(k)}) = g^{(k)} = \frac{2}{{x^{(k)}}^T B x^{(k)}} \left[Ax^{(k)} - R(x^{(k)}) B x^{(k)} \right],$$

is used to fix the direction of descent $p^{(k+1)}$ in which $R(x)$ is minimized. These directions of descent are defined by

$$p^{(1)} = -g^{(0)}, \;\; p^{(k+1)} = -g^{(k)} + \beta^{(k)} p^{(k)}, \;\; k = 1, 2, \cdots,$$

where $\beta^{(k)} = \dfrac{{g^{(k)}}^T g^{(k)}}{{g^{(k-1)}}^T g^{(k-1)}}$. The subsequent iterate $x^{(k+1)}$ along $p^{(k+1)}$ through $x^{(k)}$ is written as

$$x^{(k+1)} = x^{(k)} + \alpha^{(k+1)} p^{(k+1)}, \;\; k = 0, 1, \cdots,$$

where $\alpha^{(k+1)}$ is obtained by minimizing $R(x^{(k+1)})$,

$$R(x^{(k+1)}) = \frac{{x^{(k)}}^T A x^{(k)} + 2\alpha^{(k+1)} {p^{(k+1)}}^T A x^{(k)} + {\alpha^{(k+1)}}^2 {p^{(k+1)}}^T A p^{(k+1)}}{{x^{(k)}}^T B x^{(k)} + 2\alpha^{(k+1)} {p^{(k+1)}}^T B x^{(k)} + {\alpha^{(k+1)}}^2 {p^{(k+1)}}^T B p^{(k+1)}}.$$

A detailed explanation to get the values for $\alpha^{(k+1)}$ can be found in [8].

2.2 Preconditioned CG Method

The performance of the CG method for computing the eigenpairs of (1) can be improved by using a preconditioner [1,12]. The idea behind the PCG method is to apply the "regular" CG method to the transformed system

$$\tilde{A}\tilde{x} = \lambda\tilde{B}\tilde{x},$$

where $\tilde{A} = C^{-1}AC^{-1}, \ \tilde{x} = Cx,$ and C is nonsingular symmetric matrix. By substituting $x = C^{-1}\tilde{x}$ into (2.3), we obtain

$$R(\tilde{x}) = \frac{\tilde{x}^T C^{-1}AC^{-1}\tilde{x}}{\tilde{x}^T C^{-1}BC^{-1}\tilde{x}} = \frac{\tilde{x}^T\tilde{A}\tilde{x}}{\tilde{x}^T\tilde{B}\tilde{x}}, \tag{2.4}$$

where the matrix $\tilde{A}$ is symmetric positive definite. The transformation (2.4) leaves the stationary values of (2.3) unchanged, which are eigenvalues of (2.1), while the corresponding stationary points are obtained from $\tilde{x}_j = Cz_j, \ \ j = 1, 2, \cdots, N$. The matrix $M = C^2$ is called the preconditioner. There are a number of choices of M ranging from simple to complicated forms. In this paper, Multi-Color Block SSOR(MC-BSSOR) preconditioner is used with parallel computation aspect. The PCG algorithm for solving the smallest eigenpair with implicit preconditioning is summarized as follows.

ALGORITHM **21 The PCG method for computing the smallest eigenpair**

```
1. Compute the preconditioner $M$.
2. Choose an initial guess $x^{(0)} \neq 0$.
3. Construct the initial gradient direction $g^{(0)}$.
       Set $p^{(1)} = -g^{(0)}$ and $Mh^{(0)} = g^{(0)}$.
4. Iterate for $k = 0$ to NMAX(maximum number of iterations).
5. If $k = 0$ then set $\beta^{(k)} = 0$, otherwise compute
       $Mh^{(k)} = g^{(k)}$ and $\beta^{(k)} = \frac{{g^{(k)}}^T h^{(k)}}{{g^{(k-1)}}^T h^{(k-1)}}$.
6. Compute
       $p^{(k+1)} = -h^{(k)} + \beta^{(k)} p^{(k)}$.   (2.5)
7. Compute $\alpha^{(k+1)}$ by minimizing $R(x^{(k+1)})$.
8. Compute
       $x^{(k+1)} = x^{(k)} + \alpha^{(k+1)} p^{(k+1)}$.   (2.6)
9. Test on convergence.
```

2.3 Orthogonal Deflation-PCG Method

Although the PCG method only produces the smallest eigenpair of (1), this algorithm can also be used to evaluate a few of the smallest eigenvalues and their corresponding eigenvectors together with the aid of orthogonal deflation [3,4].

Table 1. Rate of convergence when reordering is used. h is the mesh size

	SOR	SSOR	ILU-CG
Natural Ordering	O(h)	O(h)	O($\sqrt{h}$)
Red/Black Ordering	O(h)	O(h^2)	O(h)

Following [4], the basic idea underlying orthogonal deflation-PCG method is as follows. Assume that the eigenpairs $(\lambda_i, z_i),\ i = 1, \cdots, r-1$, have been computed. To avoid convergence toward one of the computed eigenvectors $z_i,\ i = 1, \cdots, r-1$, the next initial vector $\tilde{x}_r^{(0)}$ is chosen to be B-orthogonal to $Z_{r-1} = \text{span}\{z_i \mid i = 1, \cdots, r-1\}$. And the direction vector $\tilde{p}_r^{(k)}$ is evaluated by B-orthogonalizing $p_r^{(k)}$ (in (2.5)) with respect to Z_{r-1}. Also the new approximation vector $\tilde{x}_r^{(k)}$ is evaluated by B-normalizing $x_r^{(k)}$ (in (2.6)). Now from the characterization of the eigenvectors [7]

$$R(z_r) = \min_{x \perp_B Z_{r-1}} R(x),$$

$\tilde{x}_r^{(k)}$ converges toward z_r as k increases. That is, after $z_i,\ i = 1, \cdots, r-1$ have been evaluated, z_r can be determined by minimizing $R(x)$ over the vector space which is the B-orthogonal complement to Z_{r-1}. The minimization is performed by the PCG scheme in $S2.2$.

3 Multi-coloring and Wavefront Reordering

3.1 Multi-color Reordering

Given a mesh, multi-coloring consists of assigning a color to each point so that the couplings between two points of the same color are eliminated in the discretization matrix. For example, for the 5-point Laplacian on a square with two colors in the checkerboard fashion we can remove the coupling between any two points of the same color, so that the values at all points of one color can be updated simultaneously. Similarly, four colors are needed to color the grid points of the 9-point Laplacian. However, it has been known that the convergence rate for the reordered systems often deteriorates. For the model problem SSOR and PCG method with the Red/Black ordering have a worse convergence rate than with the natural ordering, while SOR has the same rate if optimal ω is used. The table 1 contains the rates of convergence for SSOR with optimal ω, and ILU(0) PCG method with natural and red/black ordering for the 5-point Laplacian matrix [5].

The Red/Black ordering can be extended to Multi-Color ordering schemes.

3.2 Wavefront-Ordering (Level Scheduling)

Rather than pursuing the parallelisms through reordering, the wavefront technique exploits the structure of the given matrix. If the matrix comes from the discretizations of

PDEs such as by FDM or finite element method(FEM), the value of a certain node is usually depend on only the values of its neighbors. Hence, once the values of its neighbors are known that node can be updated.

Wavefront technique(or Level scheduling) is a process of finding new ordering of the nodes extracting the parallelism inherent in the matrix. This technique would work equally well for three dimensional problems as well as two dimensional problems. For references, see [10].

4 Multi-color Block SSOR Method

Multi-Coloring is a way to achieve parallelism of order N, where N is the order of the matrix. For example, it is known that for 5-point Laplacian we can order the matrix in 2-colors so that the nodes are not adjacent with the nodes with the same color. This is known as Red/Black ordering. For planar graphs maximum four colors are needed.

Blocked methods are useful in that they minimize the interprocessor communications, and increases the convergence rate as compared to point methods. SSOR is a symmetric preconditioner that is expected to perform as efficiently as incomplete Cholesky factorization combined with blocking. Instead we need to invert the diagonal block. In this paper we used the MA48 package from the Harwell library, which is a direct method using reordering strategy to reduce the fill-ins. Since MA48 type employ some form of pivoting strategy, this is expected to perform better for ill-conditioned matrices than incomplete Cholesky factorization, which does not adopt any type of pivoting strategy.

SSOR needs a ω parameter for overrelaxation. However, it is known that the convergence rate is not so sensitive to the ω parameter.

Let the domain be divided into L blocks. Suppose that we apply a multi-coloring technique, such as a greedy algorithm described in [9] to these blocks so that a block of one color has no coupling with a block of the same color. Let D_j be the coupling within the block j, and color(j) be the color of the j-th block. We denote by $U_{j,k}, k = 1, q, j < k$ and $L_{j,k}, k < j$ the couplings between the j-th color block and the k-th block.

Then, we can describe the MC-BSSOR as follows.

ALGORITHM **41 Multi-Color Block SSOR**

Let q be the total number of colors, and color(i), $i = 1, L$, be the array of the color for each block.

```
1. Choose $u_0$, and $\omega > 0$.
2. For $i > 0$ Until Convergence Do
3. For kolor = 1, q Do
4. For j = 1, L Do
5. if(color(j) = kolor) then
6. $(u_{i+1/2})_j = {D_j}^{-1}(b - \omega * \sum_{k \neq kolor}^{k=q} L_{j,k} u_{i+1/2})$.
7. endif
8. Endfor
9. For kolor = 1, q Do
10. For j = 1, L Do
11. if(color(j) = kolor) then
```

```
12. $(u_{i+1})_j = {D_j}^{-1}(u_{i+1/2} - \omega * \sum_{k \neq kolor}^{k=q} U_{j,k} u_{i+1})$.
13. endif
14. Endfor
15. Endfor
16. Endfor
```

5 Numerical Experiments

Here we test the algorithms on the following two examples.

Problem 1. Poisson equation on a square,

$$\begin{aligned} -\triangle u &= f \\ \Omega &= (0,1) \times (0,1) \\ u &= 0 \;\; on \;\; \delta\Omega \\ f &= x(1-x) + y(1-y). \end{aligned} \tag{5.7}$$

Problem 2. Elman's problem [2],

$$\begin{aligned} -(bu_x)_x - (cu_y)_y + f\,u &= g \\ \Omega &= (0,1) \times (0,1) \\ u &= 0 \;\; on \;\; \delta\Omega, \end{aligned} \tag{5.8}$$

where $b = \exp{(-xy)},\ c = \exp{(xy)},\ f = \frac{1}{(1+xy)}$, and g is such that exact solution $u = x\exp{(xy)}\sin{(\pi x)}\sin{(\pi y)}$.

For the square domain we used the Block-Row mapping, i.e, that the domain is divided into p rectangle-shaped blocks, where p is the number of the available processors. Further we assume that there is a one-to-one correspondence between the p blocks and p processors.

Tables 2-7 contain the timings for the cases with 4 preconditioners with various N. All of our test problems assume $B = I$. We used Message Passing Machine library for the interprocessor communications. We used the Block-Row mapping for the graph partitioning of the matrix. The number of colors needed is two. For the multi-coloring we have adopted the greedy heuristic as described in [9]. We have used the $epsilon$ parameter to be 10^{-6} for stopping criterion. 'X' stands for the cases with insufficient memory. As for the ω parameter we have set ω to be 1.

For the inversion of diagonal blocks in Block SSOR method, we have used the MA48 routine of the Harwell library, which adopts direct methods for sparse matrices with the reordering strategy reducing fill-ins. The cost of the MA48 is roughly proportional to L^2, where L is the size of the matrix. Since L is roughly N/p, we expect a quadratic decrease with the increasing number of processors.

The results show that for small number of processors Multi-Color ILU(0) shows the best performance, but for large number of processors the Multi-Color Block shows the best performance. In all cases, Multi-Color ILU(0) outperforms ILU(0) in the wavefront

Table 2. Problem 1 with FDM, $N = 128^2$

	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$
	CPU time				
ILU(0)/wavefront	1.6	0.8	0.8	1.1	1.6
ILU(0)/Multi-color	0.9	0.7	0.7	1.0	1.5
Point-SSOR	2.1	0.8	0.9	1.3	1.7
MC-BSSOR	3.7	1.3	0.72	0.54	0.6

Table 3. Problem 1 with FDM, $N = 256^2$

	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$
	CPU time				
ILU(0)/wavefront	5.8	3.2	2.5	2.3	3.1
ILU(0)/Multi-color	4.9	3.1	2.3	2.0	2.8
Point-SSOR	7.5	3.9	3.1	2.8	3.8
MC-BSSOR	27.2	10.7	5.0	2.2	1.4

Table 4. Problem 1 with FDM, $N = 512^2$

	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$
	CPU time				
ILU(0)/wavefront	X	X	12.5	8.6	8.2
ILU(0)/Multi-color	X	X	11.4	7.4	7.0
Point-SSOR	X	X	13.8	10.2	9.9
MC-BSSOR	X	X	34.7	14.6	7.4

order and Point SSOR preconditioners. The reason that the performance of the MC-BSSOR improves with increasing number of processors is that the block inversion cost is inversely proportional to p^2.

The results show that for small number of processors Multi-Color ILU(0) shows the best performance, but for large number of processors the Multi-Color Block shows the best performance. In all cases, Multi-Color ILU(0) outperforms ILU(0) in the wavefront order and Point SSOR preconditioners. The reason that the performance of the MC-BSSOR improves with increasing number of processors is that the block inversion cost is inversely proportional to p^2.

For N = 128^2 and 256^2 the CPUtime often increass as the number of processors increases. This is due to the communication overhead associated with the high number of processors.

Table 5. Problem 2 with FDM, $N = 128^2$

	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$
	CPU time				
ILU(0)/wavefront	2.1	1.0	1.2	1.5	2.4
ILU(0)/Multi-color	0.9	1.0	1.2	1.4	2.2
Point-SSOR	2.8	1.2	1.4	1.7	2.7
MC-BSSOR	4.3	1.6	1.0	0.7	0.8

Table 6. Problem 2 with FDM, $N = 256^2$

	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$
	CPU time				
ILU(0)/wavefront	9.3	5.1	4.1	3.8	5.1
ILU(0)/Multi-color	8.0	4.9	3.8	3.3	4.7
Point-SSOR	11.4	6.0	4.7	4.2	6.9
MC-BSSOR	29.5	12.6	6.8	3.0	2.1

Table 7. Problem 2 with FDM, $N = 512^2$

	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$
	CPU time				
ILU(0)/wavefront	X	X	18.1	12.9	12.2
ILU(0)/Multi-color	X	X	16.9	11.1	10.7
Point-SSOR	X	X	20.0	14.6	14.4
MC-SSOR	X	X	47.7	19.7	10.6

6 Conclusions

For the problems tested MC-BSSOR shows the best performance than the other preconditioners with high number of processors, while for small number of processors Multi-Color ILU(0) shows the best performance. Due to the nature of MA48 library, we expect MC-BSSOR to be *scalable* with the increasing number of processors.

References

1. Y. Cho and Y. K. Yong. A multi-mesh, preconditioned conjugate gradient solver for eigenvalue problems in finite element models. Computers and Structures, 58:575–583, 1996.
2. H. Elman. Iterative methods for large, sparse, nonsymmetric systems of linear equations. Ph. D Thesis, Yale University, 1982.
3. G. Gambolati, G. Pini, and M. Putti. Nested iterations for symmetric eigenproblems. SIAM Journal of Scientific Computing, 16:173–191, 1995.

4. G. Gambolati, F. Sartoretto, and P. Florian. An orthogonal accelerated deflation technique for large symmetric eigenproblems. Computer Methods in Applied Mechanical Engineering, 94:13–23, 1992.
5. C. -C. Jay Kuo and Tony Chan. Tow-color Fourier analysis of iterative algorithms for elliptic problems with red/black ordering. SIAM Journal of Scientific Computing, 11:767–793, 1990.
6. Sangback Ma. Comparisons of the parallel preconditioners on the CRAY-T3E for large non-symmetric linear systems. International Journal of High Speed Computing, 10:285–300, 1999.
7. B. N. Parlett. The Symmetric Eigenvalue Problem. Prentice-Hall, Englewood Cliffs, NJ, 1980.
8. A. Ruhe. Computation of eigenvalues and eigenvectors. Sparse Matrix Techniques, 130–184, V. A. Baker, ed., Springer-Verlag, Berlin, 1977.
9. Y. Saad. Highly parallel preconditioner for general sparse matrices. Recent Advances in Iterative Methods, 60:165–199, IMA Volumes in Mathematics and its Applications, G. Golub, M. Luskin and A. Greenbaum, eds, Springer-Verlag, Berlin, 1994.
10. Y. Saad. Krylov subspace methods on supercomputers. SIAM Journal of Scientific Computing, 10:1200–1232, 1989.
11. F. Sartoretto, G. Pini and G. Gambolati. Accelerated simultaneous iterations for large finite element eigenproblems. Journal of Computational Physics, 81:53–69, 1989.
12. H. R. Schwarz. Eigenvalue problems and preconditioning. International Series of Numerical Mathematics, 96:191–208, 1991.

Theoretical Relations Between Domain Decomposition and Dynamic Substructuring

Daniel J. Rixen

Delft University of Technology
Faculty of Design, Engineering and Production
Engineering Mechanics - Dynamics
Mekelweg 2, 2628 CD Delft, The Netherlands
d.j.rixen@wbmt.tudelft.nl

Abstract. Domain decomposition methods used for solving linear systems are strongly related to dynamic substructuring methods commonly used to build a reduced model. In this study we investigate some theoretical relations between the methods. In particular we discuss the conceptual similarities between the Schur Complement solvers and the Craig-Bampton substructuring techniques, both for their primal and dual form.

1 Introduction

While reduction techniques for structural dynamics models have been developed and applied for decades [1,2], fast solution techniques such as domain decomposition solvers are just becoming mature and are starting to be used in industrial applications [3]. Fast solvers and reduction techniques are two different approaches to speed up the analysis procedure in structural dynamics (as well as in other fields). Nevertheless both approaches are strongly linked and it is not always clear what the connections are.

In this paper we will expose some of the most relevant similarities and differences between model reduction and domain decomposition methods. Finally we will indicate some research challenges that will be major topics in the years to come.

2 Tearing and Assembling Substructures

Let us call Ω the (structural) domain for which a Finite Element model has been constructed. Assume now that the domain is subdivided in a number $N^{(s)}$ of non-overlapping substructures called $\Omega^{(s)}$ such that every node belongs to one and only one substructures except for those on the interface boundaries. The linear dynamic behavior of each substructure $\Omega^{(s)}$ is governed by the local equilibrium equations

$$M^{(s)}\ddot{u}^{(s)} + K^{(s)}u^{(s)} = f^{(s)} + g^{(s)} \qquad s = 1, \dots N_s \tag{2.1}$$

where $M^{(s)}$ and $K^{(s)}$ are the substructure mass and stiffness matrices, $u^{(s)}$ are the local dof, $f^{(s)}$ the external loads applied to the substructure and $g^{(s)}$ the internal forces on the interfaces between substructures that ensure compatibility. Damping will not be discussed here for simplicity.

2.1 Primal Assembly

The substructures (commonly called sub-domains in domain decomposition terminology) can be interpreted as macro-element: the local degrees of freedom $u^{(s)}$ are related to a global set of assembled degrees of freedom u_a by

$$u^{(s)} = L^{(s)} u_a \tag{2.2}$$

where $L^{(s)}$ is a Boolean matrix. Substituting the compatibility condition (2.2), the local equilibrium equations (2.1) can be assembled as

$$M_a \ddot{u}_a + K_a u_a = f_a(t) \tag{2.3}$$

where $\ddot{u}_a$ is the second-order time derivative of u_a and

$$M_a = \sum_{s=1}^{N_s} L^{(s)^T} M^{(s)} L^{(s)} \qquad K_a = \sum_{s=1}^{N_s} L^{(s)^T} K^{(s)} L^{(s)} \tag{2.4}$$

are the assembled mass and stiffness matrices. Note that the interface forces $g^{(s)}$ cancel out when assembled on the interface.

2.2 Dual Assembly

Let us once more consider the decomposed equations (2.1) and explicitly express the interface forces as unknowns determined by the interface compatibility requirement (2.2):

$$\begin{cases} M^{(s)} \ddot{u}^{(s)} + K^{(s)} u^{(s)} + \begin{bmatrix} b^{(s)^T} \lambda \\ 0 \end{bmatrix} = f^{(s)} \\ \sum_{s=1}^{N_s} b^{(s)} u_b^{(s)} = 0 \end{cases} \tag{2.5}$$

where $b^{(s)}$ are signed Boolean matrices and where $b^{(s)^T} \lambda$ represent the interconnecting forces between substructures. The description (2.5) is in fact a mixed approach where primal (displacements) are used inside the substructures while dual quantities (forces) describe the interface problem. In the domain decomposition world, the approach described in (2.5) is called dual. Note that those equations are exactly equivalent to the assembled system (2.3) since they express the same local equilibrium and enforce the same interface compatibility. However (2.3) is in fact a primal approach since the interface problem is described in terms of displacements only and therefore both approaches will lead to different solution techniques. The dual (or mixed formulation) (2.5) can be written in the block format

$$\begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{u} \\ \lambda \end{bmatrix} + \begin{bmatrix} K & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} u \\ \lambda \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix} \tag{2.6}$$

More details on dual assembly and its connection to primal assembly can be found in [3,4].

3 Dynamic Substructuring and Domain Decomposition Solvers

Structural models for industrial applications are most often very large (of the order of several millions of degrees of freedom) because the models are often inherited from static models where the meshing is fine, or because complex meshes are needed in order to represent the geometrical and topological complexity of the structure.

Solving dynamical problems requires solving a large number of static-like problems, either in the inverse-iteration loops of the eigensolvers or when stepping through the time or the frequency in transient or harmonic analysis. For a vast majority of applications, the dynamical behavior relevant for the analysis at hand involves simple global deformation modes and therefore could be represented with a simple discretization whereas the models at hand are very complex and refined. Therefore reducing the models before solving them is often a necessary step. This does not mean that the mesh is going to be coarsened (this would be a very tedious if not impossible task), but rather the dynamic behavior of the model will be represented by a small number of well chosen modes. One approach is to use methods based on global modes such as in the Rayleigh-Ritz methods [5] or in the Proper Orthogonal decomposition technique. Another approach, better suited for parallel computation and in industrial projects consists in reducing every submodel such as in the dynamic substructuring techniques. This will be discussed in section 3.1

Another way to tackle the challenge of solving many large static-like problems such as encountered in dynamics consists in applying highly efficient parallel solvers derived from domain decomposition techniques. They will be shortly described in section 3.2

3.1 Primal and Dual Substructuring

An efficient way to build reduced models consists in building reduction bases for each substructures. We will shortly lay out the primal and dual approaches to reduction.

In the primal approach, each substructure is considered as being excited through its interface degrees of freedom and therefore the interior degrees of freedom are governed by the equilibrium equations

$$M_{ii}^{(s)}\ddot{u}_i^{(s)} + K_{ii}^{(s)}u_i^{(s)} = f_i^{(s)} - K_{ib}^{(s)}u_b^{(s)} - M_{ib}^{(s)}\ddot{u}_b^{(s)} \tag{3.7}$$

Therefore the natural reduction basis for the internal degrees of freedom $u_i^{(s)}$ is given by the eigenmodes associated with the matrices $M_{ii}^{(s)}$ and $K_{ii}^{(s)}$, namely the eigenmodes of the substructures when the interface is fixed. An efficient reduction basis is then

$$u_a = \begin{bmatrix} u_b \\ u_i^{(1)} \\ \vdots \\ u_i^{(N_s)} \end{bmatrix} \simeq \begin{bmatrix} I & 0 & \cdots & 0 \\ \Psi^{(1)}L_b^{(1)} & \Phi^{(1)} & & 0 \\ \vdots & & \ddots & \\ \Psi^{(N_s)}L_b^{(N_s)} & 0 & & \Phi^{(N_s)} \end{bmatrix} \begin{bmatrix} u_b \\ \eta^{(1)} \\ \vdots \\ \eta^{(N_s)} \end{bmatrix} = T_{CB} \begin{bmatrix} u_b \\ \eta^{(1)} \\ \vdots \\ \eta^{(N_s)} \end{bmatrix} \tag{3.8}$$

where

$$\Psi^{(s)} = -K_{ib}^{(s)}K_{ii}^{(s)^{-1}} \tag{3.9}$$

are static response modes and where $\Phi^{(s)}$ are $n_i^{(s)} \times n_\phi^{(s)}$ matrices which columns contain the first $n_\phi^{(s)}$ free vibration modes of the substructure clamped on its interface. The reduced matrices are then

$$\bar{K}_{CB} = T_{CB}^T K_a T_{CB} = \begin{bmatrix} S_{bb} & & & 0 \\ & \Lambda^{(1)^2} & & \\ & & \ddots & \\ 0 & & & \Lambda^{(N_s)^2} \end{bmatrix}$$

$$\bar{M}_{CB} = T_{CB}^T M_a T_{CB} = \begin{bmatrix} M_{bb}^* & L_b^{(s)^T} M_{b\phi}^{(1)} & \cdots & L_b^{(s)^T} M_{b\phi}^{(N_s)} \\ M_{\phi b}^{(1)} L_b^{(s)} & I & & 0 \\ \vdots & & \ddots & \\ M_{\phi b}^{(N_s)} L_b^{(s)} & 0 & & I \end{bmatrix}$$

where $\Lambda^{(s)^2}$ are diagonal matrices of the $n_\phi^{(s)}$ local eigenfrequencies and where S_{bb} and M_{bb}^* are the statically condensed stiffness and mass matrices. This approach is known as the Craig-Bampton reduction [6]. Many variants have been proposed based on this idea. See for instance [1] for an comprehensive overview.

In the dual approach, each substructure is considered as being excited through the interface forces as described by (2.5). The substructure response can therefore be approximated as a static response plus a dynamic contribution related to the eigenmodes of the substructure when its interface is free:

$$u^{(s)} \simeq u_{stat}^{(s)} + \Theta^{(s)} \eta^{(s)} \tag{3.10}$$

where

$$u_{stat}^{(s)} = -K^{(s)^+} B^{(s)^T} \lambda + \sum_{i=1}^{m^{(s)}} \mathrm{R}_i^{(s)} \alpha_i^{(s)} \tag{3.11}$$

where $K^{(s)^+}$ is the inverse of $K^{(s)}$ when there are enough boundary conditions to prevent the substructure from floating when its interface with neighboring domains is free [5]. If a substructure is floating, $K^{(s)^+}$ is a generalized inverse of $K^{(s)}$ and $\mathrm{R}^{(s)}$ is the matrix having as column the corresponding rigid body modes. $\alpha_i^{(s)}$ are amplitudes of the local rigid body modes. The dynamic contribution is approximated in (3.10) by a small number $k^{(s)}$ of local eigenmodes forming the reduced dynamic basis $\Theta^{(s)}$. The local degrees of freedom and the Lagrange multipliers (interconnecting forces) can thus be approximated by [7]

$$\begin{bmatrix} u \\ \lambda \end{bmatrix} = T_{dual} \begin{bmatrix} \alpha^{(1)} \\ \eta^{(1)} \\ \vdots \\ \alpha^{(N_s)} \\ \eta^{(N_s)} \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathrm{R}^{(1)}\ \Theta^{(1)} & & 0 & -G_{res}^{(1)} B^{(1)^T} \\ & \ddots \ \ddots & & \vdots \\ 0 & & \mathrm{R}^{(N_s)}\ \Theta^{(N_s)} & -G_{res}^{(N_s)} B^{(N_s)^T} \\ 0 & \cdots & 0 & I \end{bmatrix} \begin{bmatrix} \alpha^{(1)} \\ \eta^{(1)} \\ \vdots \\ \alpha^{(N_s)} \\ \eta^{(N_s)} \\ \lambda \end{bmatrix} \tag{3.12}$$

where the residual flexibilities are defined as

$$G_{res}^{(s)} = K^{(s)^+} - \sum_{r=1}^{n_\theta^{(s)}} \frac{\theta_r^{(s)} \theta_r^{(s)^T}}{\Lambda_r^{(s)^2}} \tag{3.13}$$

Finally, applying the reduction transformation (3.12) to the dual form (2.5), one gets the so-called dual Craig-Bampton reduced system [7]:

$$\tilde{M} \begin{bmatrix} \vdots \\ \ddot{\alpha}^{(s)} \\ \ddot{\eta}^{(s)} \\ \vdots \\ \ddot{\lambda} \end{bmatrix} + \tilde{K} \begin{bmatrix} \vdots \\ \alpha^{(s)} \\ \eta^{(s)} \\ \vdots \\ \lambda \end{bmatrix} = T_{dual}^T f \tag{3.14}$$

with the reduced hybrid matrices

$$\tilde{M} = T_{dual}^T \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} T_{dual} \tag{3.15}$$

$$\tilde{K} = T_{dual}^T \begin{bmatrix} K & B^T \\ B & 0 \end{bmatrix} T_{dual} \tag{3.16}$$

3.2 Domain Decomposition

Domain decomposition methods (DD) can be seen as fast solvers. For a static problem

$$K_a u_a = f_a \tag{3.17}$$

primal DD approaches consists in first factorizing the substructure internal operators in order to write the interface problem as the statically condensed problem

$$S_{bb} u_b = f_a^* \tag{3.18}$$

An efficient manner to solve (3.17) on parallel processing machines is then found by solving (3.18) iteratively (usually with a Conjugate Gradient scheme). Hence, at every iteration an new estimate of the interface displacement is found and, after solving the local problems for the internal degrees of freedom, the interface equilibrium residual is computed.

In a dual DD approach [4], the local degrees of freedom $u^{(s)}$ are condensed out from the dual static problem

$$\begin{cases} K^{(s)}u^{(s)} + \begin{bmatrix} b^{(s)^T}\lambda \\ 0 \end{bmatrix} = f^{(s)} \\ \displaystyle\sum_{s=1}^{N_s} b^{(s)}u_b^{(s)} = 0 \end{cases} \tag{3.19}$$

The dual interface problem can then be written in terms of λ alone. Solving for the interface forces λ iteratively then yields an efficient parallel solver. At every iteration an estimate of the interface forces is found. Then the corresponding local solutions are computed and the compatibility error on the interface is found.

A mechanical description of the primal and dual DD approaches as well as a description of the different variants of the methods for instance for multiple right hand sides can be found in the review paper [8].

3.3 Connections and Differences

Obviously the substructuring techniques and the domain decomposition solvers are strongly related since they all originate from the same primal or dual decomposed description of the problem.

In order to understand the major connection between substructuring and domain decomposition solvers, let us assume that inverse iteration methods are applied on the primal or dual Craig-Bampton reduced systems (3.10) and (3.15). An inverse iteration step in the primal method corresponds to solving a static-like problem of the form (3.18) and thus corresponds to the primal DD formulation. Hence applying a primal DD iterative scheme to solve the iterface problem for u_b thus corresponds to building successive approximations for the interface degrees of freedom u_b.

If we now write an inverse iteration step for the dual Craig-Bampton reduced model (3.15) it is readily seen that the static problem to solve corresponds to the dual interface problem as solved in the dual DD method.

Hence one can conclude that the main differences between solving dynamic problems based on reduced Craig-Bampton models or applying DD methods directly on the non-reduced system are that

- In the Craig-Bampton models, the dynamical behavior of the substructures is approximated by a reduced modal basis, whereas when applying DD methods on the full model no reduction is performed on the substructure.
- When applying DD techniques to solve the interface problem iteratively, one automatically builds an approximation basis for the interface degrees of freedom or forces, whereas in the standard Craig-Bampton reduction procedures interface variables are not reduced (note that reduction techniques that *a priori* define reduction basis for the interface problem have also been proposed).

Hence it would be equivalent to use DD solvers to solve the interface problems arising in reduction procedures, or to apply DD solvers where the local problems are approximated by reduced modal basis.

The combination of reduction of substructures and iterative solution of interface problems appears to be a very attractive idea to solve large dynamical problems efficiently. Moreover such an approach is naturally parallel and has therefore more potential for efficient implementation on supercomputers than the fast but inherently sequential multi-level substructuring methods [9].

4 Conclusion

In this presentation, we show the links between reduction techniques based on dynamic substructuring and domain decomposition solution techniques. It appears that combining reduction techniques for the local dynamic behavior of substructures together with iterative solution of the interface problems opens new opportunities to build fast and parallel solvers for structural dynamics. Future research will investigate the possibility of such approaches particularly when applied to industrial problems.

References

1. Roy R. Craig. Coupling of substructures for dynamic analyses: an overview. In *Structures, Structural Dynamics and Material Conference*, Atlanta, April 3-6 2000. 41st AIAA/ASME/ASCE/AHS/ASC. AIAA-2000-1573.
2. S. H. Fransen. Data Recovery Methodologies for Reduced Dynamic Substructure Models with Internal Loads. *Am. Inst. Aero. Astro. J.*, 42(10):2130–2142, 2004.
3. P. Gosselet and C. Rey and D.J. Rixen On the initial estimate of interface forces in FETI methods. *Int. J. Num. Meth. Eng.*, 192:2749–2764, 2003.
4. C. Farhat and F. X. Roux. Implicit parallel processing in structural mechanics. *Computational Mechanics Advances*, 2(1):1–124, 1994. North-Holland.
5. M. Géradin and D. Rixen. *Mechanical Vibrations. Theory and Application to Structural Dynamics*. Wiley & Sons, Chichester, 2d edition, 1997.
6. R.R. Craig and M.C.C. Bampton. Coupling of substructures for dynamic analysis. *Am. Inst. Aero. Astro. J.*, 6(7):1313–1319, July 1968.
7. Daniel Rixen. A Dual Craig-Bampton Method for Dynamic Substructuring. *J. Comp. App. Math.*, 168 (1-2):383–391, 2004.
8. Daniel Rixen. *Encyclopedia of Vibration*, chapter Parallel Computation, pages 990–1001. Academic Press, 2002. isbn 0-12-227085-1.
9. Jeffrey K. Bennighof and R. B. Lehoucq. An automated multilevel substructuring method for eigenspace computation in linear elastodynamics. *SIAM J. Sc. Computing*, 2002. http://www.cs.sandia.gov/ rlehoucq/papers.html.

Model Order Reduction for Large Scale Engineering Models Developed in ANSYS

Evgenii B. Rudnyi and Jan G. Korvink

IMTEK, Institute of Microsystem Technology
Freiburg University
Georges-Köhler-Allee, 103
D-79110, Freiburg, Germany
{rudnyi,korvink}@imtek.de
http://www.imtek.uni-freiburg.de/simulation/

Abstract. We present the software `mor4ansys` that allows engineers to employ modern model reduction techniques to finite element models developed in ANSYS. We focus on how one extracts the required information from ANSYS and performs model reduction in a C++ implementation that is not dependent on a particular sparse solver. We discuss the computational cost with examples related to structural mechanics and thermal finite element models.

1 Introduction

The model order reduction of linear large-scale dynamic systems is already quite an established area [1]. In many papers (see references in [2]), advantages of model reduction have been demonstrated for a variety of scientific and engineering applications. In the present work, we focus on how engineers can combine this technique with existing commercial finite element software in order to

– Speed up a transient or harmonic analysis,
– Generate automatically compact models for system-level simulation,
– Incorporate finite element packages during the design phase.

Model reduction is conventionally applied to a large-scale dynamic system of the first order as follows

$$\begin{aligned} E\dot{\boldsymbol{x}} &= A\boldsymbol{x} + B\boldsymbol{u} \\ \boldsymbol{y} &= C\boldsymbol{x} \end{aligned} \tag{1.1}$$

where A and E are system matrices, B is the input matrix, C is the output matrix. The aim of model reduction is to generate a low-dimensional approximation to (1.1) in a similar form

$$\begin{aligned} E_r\dot{\boldsymbol{z}} &= A_r\boldsymbol{z} + B_r\boldsymbol{u} \\ \boldsymbol{y} &= C_r\boldsymbol{z} \end{aligned} \tag{1.2}$$

that describes well the dependence of the output vector $\boldsymbol{y}$ on the input vector $\boldsymbol{u}$ and so that, at the same time, the dimension of the reduced state vector $\boldsymbol{z}$ is much less than the dimension of the original state vector $\boldsymbol{x}$.

After discretization in space of the partial differential equations describing a user model, a finite element package generally produces a system of ordinary differential equations. At this stage, it is possible to directly apply modern model reduction methods [1]. However, the extraction of the system matrices from a commercial package happens not to be straightforward and here we share our experience on how it can be done with ANSYS [3].

We have chosen the Matrix Market format [4] to represent the reduced model (1.2). We suppose that its simulation will be done in another package, such as Matlab or Mathematica. Functions to work with the reduced model in Mathematica are available at the IMTEK Mathematica Supplement at `http://www.imtek.uni-freiburg.de/simulation/mathematica/IMSweb/`.

The system matrices are high-dimensional and sparse. As a result, the implementation of a model reduction algorithm usually depends on a particular sparse solver and a storage scheme for sparse matrices. We discuss a C++ interface that allows us to isolate the model reduction and sparse solvers completely for negligible overhead.

Finally, we analyse the computation cost and give the performance results for a few ANSYS models. The comparison of the accuracy of reduced models in respect to the original ANSYS models is given elsewhere [5].

2 mor4ansys

The developed software [6] comprises two almost independent modules (see Fig. 1). The first reads a binary ANSYS file and assembles a dynamic system in the form of Eq (1.1) for first order systems or

$$\begin{aligned} M\ddot{\boldsymbol{x}} + E\dot{\boldsymbol{x}} + K\boldsymbol{x} &= B\boldsymbol{u} \\ \boldsymbol{y} &= C\boldsymbol{x} \end{aligned} \tag{2.3}$$

for second order systems, where M, E and K are the three system matrices. The second module applies the model reduction algorithm to Eq (1.1) or (2.3), that is, it finds a low-dimensional basis V so that the approximation

$$\boldsymbol{x} = V\boldsymbol{z} + \boldsymbol{\epsilon} \tag{2.4}$$

allows us to reproduce the transient behaviour of the original state vector within the error margin $\boldsymbol{\epsilon}$.

After that, the original equations are projected to the subspace found, for example for Eq (1.2) we have $E_r = V^T EV$, $A_r = V^T AV$, $B_r = V^T B$, $C_r = CV$.

We support three methods to treat second-order systems. When the damping matrix is modeled as Rayleigh damping $E = \alpha M + \beta K$, the method from Ref [7] allows us to preserve the coefficients α and β as parameters in the reduced model. In the general case, one can choose between the transformation to a first-order system, and second order Arnoldi algorithm (SOAR) [8].

The software can also read, as well as write, the matrices for the original system in the Matrix Market format [4]. A number of model reduction benchmarks has been obtained from ANSYS by means of `mor4ansys` [9].

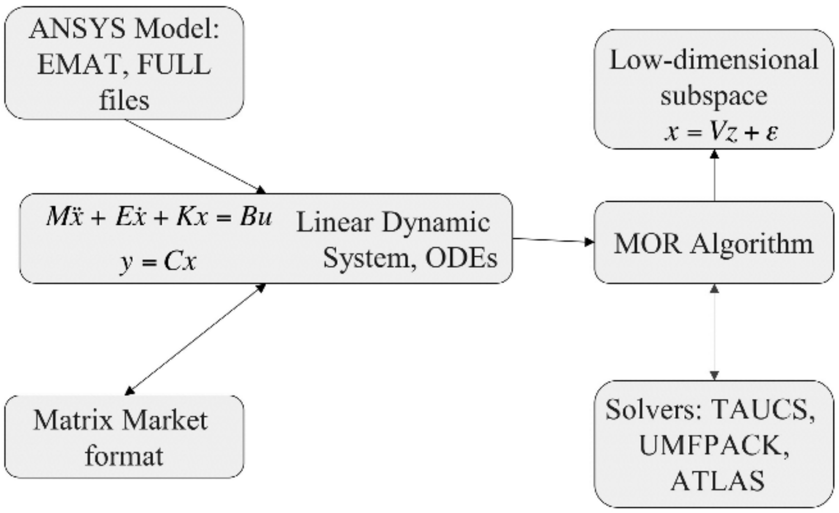

ANSYS Model: EMAT, FULL files
$M\ddot{x} + E\dot{x} + Kx = Bu$
$y = Cx$
Linear Dynamic System, ODEs
Matrix Market format
Low-dimensional subspace
$x = Vz + \varepsilon$
MOR Algorithm
Solvers: TAUCS, UMFPACK, ATLAS

Fig. 1. `mor4ansys` block-scheme

2.1 Interfacing with ANSYS

The development of the first module happen to be rather difficult because most users of a commercial finite element package do not need the capability to extract the dynamics system in the form of Eq (1.1) or (2.3) and, as a result, this is not a trivial operation.

ANSYS is a huge package and its behavior is not completely consistent. For example, the information described below is not applicable for the fluid dynamics module FLOTRAN.

Our software reads the binary `EMAT` file with element matrices in order to assemble global system matrices. The file format is documented and ANSYS supplies a library of Fortran subroutines to work with it [10]. An example of how one can use them can be found in the `mor4ansys` code [6]. ANSYS has a special command, called a partial solve `PSOLVE`, with which one can evaluate element matrices for a given state vector without going through the real solution stage. This allows us to generate an `EMAT` file efficiently for a given model. However, it was necessary to overcome the following problems:

- The `EMAT` file does not contain the information about either Dirichlet boundary conditions or equation constraints. They should be extracted separately.
- The `EMAT` file has a contribution to the load vector from element matrices only. If nodal forces or accelerations are used to apply the load, this information should also be extracted individually.
- It is necessary to assemble the global matrices from the element matrices.

During the solution phase, ANSYS can write a binary `FULL` file with the assembled system matrices. When we started the development with ANSYS 5.7, this file did not contain the load vector (input matrix). Since then there have been many changes. Since

ANSYS 6.0 the `FULL` file maintains all the original matrices, the load vector, the Dirichlet and equation constraints in the file. ANSYS 8.0 allows us to make the assembly only and write the `FULL` file without a real solution phase (equivalent to a partial solution with `EMAT`). One can now also dump the information from the `FULL` file in the Harwell-Boeing matrix format. Hence, since ANSYS 8.0, it is possible to use the `FULL` file efficiently. However, depending on the analysis type the `FULL` file may contain not the original stiffness matrix, but rather, a linear combination of system matrices instead.

In the current version of `mor4ansys`, the `EMAT` file is employed as the main source to build Eq (1.1) or (2.3). Additional information on the Dirichlet and equation constraints and nodal forces is written in the form of text files by means of ANSYS macros we have developed. The `FULL` file can be used to extract the load vector when otherwise this is difficult, for example, as in the case when the acceleration load is used.

ANSYS cannot write several load vectors into the `FULL` and `EMAT` files. When multiple-input is to be preserved in Eq (1.1) or (2.3), a user should for each input:

– Delete the previously applied load,
– Apply a new load,
– Generate matrices.

In order to ease this process, the second strategy is also allowed when a user does not have to delete the previous load. In this case, each new load vector contains all the previous vectors and `mor4ansys` corrects them at the end of the first phase.

2.2 Running the Model Reduction Algorithm

The Krylov subspaces allow us to obtain a low-dimensional subspace basis for (2.4) with excellent approximating properties by means of a very efficient computation [11,8]. The current version of `mor4ansys` implements the block Arnoldi algorithm [11] in order to support multiple inputs, the block size being equal to the number of inputs.

Each step of an iterative Krylov subspace algorithm requires us to compute a matrix-vector product, for example, for the first-order system

$$A^{-1}E\boldsymbol{h} \tag{2.5}$$

where $\boldsymbol{h}$ is some vector. The system matrices are high-dimensional and sparse and one does not compute A^{-1} explicitly. The only feasible solution is to solve a linear system of equations for each step as follows

$$A\boldsymbol{g} = E\boldsymbol{h} \tag{2.6}$$

This constitutes the main computational cost up to the order of the reduced system 30. Later on, the additional cost associated with the orthogonolization process can be also added.

There are many sparse solvers as well as many storage schemes for sparse matrices. Our goal was to implement a model reduction algorithm in a way that does not depend on a particular solver. In addition, we wanted to change solvers at run-time, that is, to allow for run-time polymorphism. As a result, we have chosen the virtual function

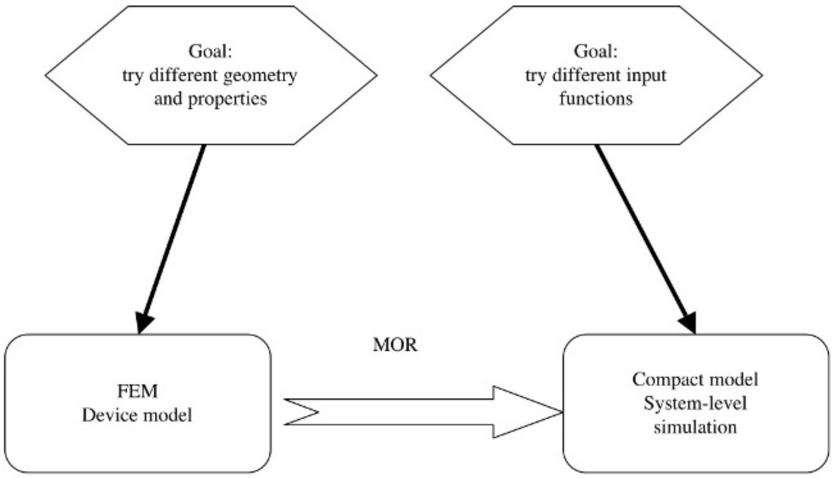

Goal:
try different geometry and properties
Goal:
try different input functions
FEM
Device model
MOR
Compact model
System-level simulation

Fig. 2. Use of model reduction during design and system-level simulation

mechanism, as its overhead is negligible in our case when the operations by themselves are computationally intensive.

Our approach is similar to that in the PETs [12] and Trinilos [13] libraries. The abstract interface is written in terms of relatively low-level functions, as the goal was to cover many different scenarios. The vectors are represented by continuous memory, as they are dense in the case of the Krylov subspaces.

At present, the direct solvers from the TAUCS [14] and UMFPACK [15,16] libraries are supported. The ATLAS library [17] has been used to generate the optimized BLAS. We have found that for many ANSYS models up to 500 000 degrees of freedom the modern direct solvers are quite competitive as the matrix factor fits within 4 Gb of RAM. This allows us to reuse the factorization and achieve good performance.

3 Computational Cost of Model Reduction

We have experimentally observed that for many ANSYS models a reduced model of order 30 is enough to accurately represent the original high-dimensional system [5]. Hence, for simplicity we limit the analysis of the computational cost to this case.

The simulation time of the reduced system comprising 30 equations is very small and we can neglect it. Therefore, for the case when several simulations with different input functions are necessary (the system-level simulation case), the advantage of model reduction is out of the question.

Yet, during the design phase, a reduced model should be generated each time when a user changes the geometry or material properties of the original model. In this case, a reduced model might be used just once. Nevertheless, the model reduction time can be smaller than the simulation time of the original system even in this case. These two different situations are shown in Fig. 2. Below we consider the second case.

Table 1. Computational times on Sun Ultra-80 with 4 Gb of RAM in seconds

dimension	nnz	stationary solution in ANSYS 7.0	stationary solution in ANSYS 8.0	factoring in TAUCS	generation of the first 30 vectors
4 267	20 861	0.87	0.63	0.31	0.59
11 445	93 781	2.1	2.2	1.3	2.7
20 360	265 113	16	15	12	14
79 171	2 215 638	304	230	190	120
152 943	5 887 290	130	95	91	120
180 597	7 004 750	180	150	120	160
375 801	15 039 875	590	490	410	420

Let us assume that a direct solver is applicable and the dimension of 30 for the reduced system is sufficient. Then the model reduction time is equal to the time of factoring A in Eq (2.5) and the time required for 30 back substitution steps in Eq (2.6). Table 1 presents computational times for seven ANSYS models where the system matrices are symmetric and positive definite. The first four rows correspond to thermal simulations [18] and the last three to structural mechanics of a bond wire [7].

Each case is specified by its dimension and the number of non zero elements in the stiffness matrix. The time of a stationary solution in ANSYS is given as a reference point. Note that the real simulation time in ANSYS required for the stationary solution is larger than in Table 1 as it includes reading/writing files as well as some other operations. After that is listed the time to factor a matrix by means of a multifrontal solver from the TAUCS library [14] and the time to generate the first 30 vectors. The latter is dominated by the solution of Eq (2.6) by means of back substitution. As the difference to generate the first and thirtieth vectors was less than 10-20%, we can say that the orthogonalization cost was relatively small.

Note that the TAUCS multifrontal solver is even faster than the ANSYS solver. The total time to generate a reduced model is about twice more than that for the stationary solution. At the same time, the reduced model can accurately reproduce any transient and harmonic simulation of the original models within a reasonable frequency range.

The simulation time of a harmonic analysis is the product of solution time for a complex linear system by the number of frequencies needed. The matrix factor cannot be re-used as the linear system to solve depends on frequency. The solution time for a complex linear system is about twice more expensive. Hence model reduction allows us to save simulation time by a factor close to the number of frequencies at which the harmonic response is required. For example, if it is necessary to estimate the transfer function at ten frequencies, then the model reduction plus the simulation of the reduced system is roughly ten times faster than the simulation of the original system.

For the transient simulation, the situation is more difficult to analyse as this depends on the integration strategy. In principle, it is possible to say that the model reduction time above is equivalent to 30 equally spaced timesteps as in this case the same strategy with the re-use of the matrix factor can be applied. However, in our experience, in order to achieve accurate integration results for the examples in Table 1, one either needs at least 600 equally-spaced timesteps or one needs to use adaptive integration schemes where the factor re-use is not possible. In both cases, model reduction plus simulation of the reduced system was more than ten times faster. This shows that model reduction can also be viewed as a fast solver and can be employed even during the optimization phase.

4 Conclusions

We have shown that in the case of the linear dynamics systems (1.1) and (2.3) modern model reduction techniques can speed up finite element transient and harmonic simulation significantly. For nonlinear systems, there are promising theoretical results in the case of polynomial type nonlinearity [19]. Yet, in the nonlinear case in addition to many theoretical problems, it happens that extracting a nonlinear system (1.1) or (2.3) from a commercial finite element tool is a challenge by itself.

Acknowledgment

ANSYS models of the microthruster and the bonded wire have been made by T. Bechtold and J. Lienemann respectively. We would also like to acknowledge an anonymous reviewer for the very helpful comments and suggestions to improve the paper. Partial funding by the DFG project MST-Compact (KO-1883/6), the Italian research council CNR together with the Italian province of Trento PAT, the European Union (grant EU IST-1999-29047, Micropyros) and an operating grant of the University of Freiburg is gratefully acknowledged.

References

1. A. C. Antoulas, D. C. Sorensen. Approximation of Large-Scale Dynamical Systems: An overview. *Applied Mathematics & Computer Science*, 11(5):1093–1121, 2001.
2. E. B. Rudnyi, J. G. Korvink. Automatic Model Reduction for Transient Simulation of MEMS-based Devices. *Sensors Update*, 11:3–33, 2002.
3. ANSYS, ANSYS Inc. http://www.ansys.com/
4. R. F. Boisvert, R. Pozo, K. A. Remington. The Matrix Market Exchange Formats: Initial Design. *NIST Interim Report 5935*, 1996. http://math.nist.gov/MatrixMarket/
5. E. B. Rudnyi, J. G. Korvink. Model Order Reduction of MEMS for Efficient Computer Aided Design and System Simulation. In *Sixteenth International Symposium on Mathematical Theory of Networks and Systems*, Belgium, July 5-9, 2004. Minisymposium TA8: Issues in model reduction of large-scale systems.
6. E. B. Rudnyi, J. G. Korvink. mor4ansys (version 1.6): Compact Behavioral Models from ANSYS by Means of Model Order Reduction. *User Manual*, 2004. http://www.imtek.uni-freiburg.de/simulation/mor4ansys/
7. E. B. Rudnyi, J. Lienemann, A. Greiner, and J. G. Korvink. mor4ansys: Generating Compact Models Directly from ANSYS Models. In *Technical Proceedings of the 2004 Nanotechnology Conference and Trade Show*, Nanotech 2004, March 7-11, 2004, Bosten, Massachusetts, USA.

8. Z. J. Bai, K. Meerbergen, Y. F. Su. Arnoldi methods for structure-preserving dimension reduction of second-order dynamical systems. In: P. Benner, G. Golub, V. Mehrmann, D. Sorensen (eds), *Dimension Reduction of Large-Scale Systems, Lecture Notes in Computational Science and Engineering.* Springer-Verlag, Berlin/Heidelberg, Germany, 2005.
9. J. G. Korvink, E. B. Rudnyi. Oberwolfach Benchmark Collection. In: P. Benner, G. Golub, V. Mehrmann, D. Sorensen (eds), *Dimension Reduction of Large-Scale Systems, Lecture Notes in Computational Science and Engineering.* Springer-Verlag, Berlin/Heidelberg, Germany, 2005. http://www.imtek.uni-freiburg.de/simulation/benchmark/
10. Guide to Interfacing with ANSYS, ANSYS Inc. 2001.
11. R. W. Freund. Krylov-subspace methods for reduced-order modeling in circuit simulation. *Journal of Computational and Applied Mathematics*, 123: 395–421, 2000.
12. S. Balay, V. Eijkhout, W. D. Gropp, L. C. McInnes, B. F. Smith. Efficient Management of Parallelism in Object Oriented Numerical Software Libraries. In E. Arge, A. M. Bruaset, H. P. Langtangen (eds), *Modern Software Tools in Scientific Computing*, Birkhäuser Press, 163–202, 1997.
http://www-unix.mcs.anl.gov/petsc/petsc-2/
13. M. Heroux, R. Bartlett, V. Howle, R. Hoekstra, J. Hu, et al. An Overview of Trilinos. *Sandia National Laboratories report* SAND2003-2927, 2003. http://software.sandia.gov/trilinos/
14. V. Rotkin, S. Toledo. The design and implementation of a new out-of-core sparse Cholesky factorization method. *ACM Transactions on Mathematical Software*, 30: 19–46, 2004.
http://www.tau.ac.il/ stoledo/taucs/
15. T. A. Davis. Algorithm 832: UMFPACK V4.3, an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2): 196–199, 2004.
http://www.cise.ufl.edu/research/sparse/umfpack/
16. T. A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2): 165–195, 2004.
17. R. C. Whaley, A. Petitet, J. Dongarra. Automated Empirical Optimization of Software and the ATLAS project. *Parallel Computing*, 27(1-2): 3-35, 2001.
http://math-atlas.sourceforge.net/
18. J. Lienemann, E. B. Rudnyi, J. G. Korvink. MST MEMS model order reduction: Requirements and Benchmarks. Submitted to *Linear Algebra and its Applications*, 2004.
19. J. R. Phillips. Projection-based approaches for model reduction of weakly nonlinear, time-varying systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22:171–187, 2003.

Rational Krylov for Large Nonlinear Eigenproblems

Axel Ruhe[⋆]

Department of Numerical Analysis and Computer Science
Royal Institute of Technology, SE-10044 Stockholm, Sweden
ruhe@kth.se

Abstract. Rational Krylov is an extension of the Lanczos or Arnoldi eigenvalue algorithm where several shifts (matrix factorizations) are used in one run. It corresponds to multipoint moment matching in model reduction. A variant applicable to nonlinear eigenproblems is described.

1 Introduction, Nonlinear Eigenvalues

We are interested in the eigenvalue problem

$$A(\lambda)x = 0 \tag{1.1}$$

linear in the vector x but nonlinear in the eigenvalue parameter λ. Typical examples are mechanical systems with viscous damping or nonlinear materials. We seek values of λ where the matrix $A(\lambda)$ is singular.

For reasonably large linear eigenproblems, a shift and invert spectral transformation is the algorithm of choice, see the collection [2]! A Rational Krylov approach makes it possible to use several shifts (poles) in one run [10]. This is of special interest in a model reduction context, where it corresponds to multipoint moment matching, see reports by Skoogh [12] and Olsson [8]!

For moderate dimension n, the nonlinear eigenproblem (1.1) is treated by varying iterative algorithms, see the early work by Kublanovskaya for a QR Newton algorithm [7]. The evident idea of solving a sequence of linear eigenproblems, studied in [9], was shown to have significant advantages.

When the dimension is large, methods based on solving nonlinear problems projected in subspaces expanded by Krylov or Jacobi Davidson, is reported in works by Voss and collaborators [14,3,**?**].

We have developed a nonlinear Rational Krylov algorithm, see the preliminary report [11] and the thesis works by Hager [5] and Jarlebring [6]. Let me describe this approach!

Approximate $A(\lambda)$ by linear Lagrange interpolation between a *shift* μ and a *pole* σ and get

$$A(\lambda) = \frac{\lambda-\mu}{\sigma-\mu}A(\sigma) + \frac{\lambda-\sigma}{\mu-\sigma}A(\mu) + (\lambda-\sigma)(\lambda-\mu)R(\lambda)\,. \tag{1.2}$$

[⋆] Talk given at PARA 04, Copenhagen June 21, 2004. Partial support given by VR, the Swedish Research Council.

If we disregard the second order term with $R(\lambda)$, we predict singularity of $A(\lambda)$ at λ satisfying

$$[(\lambda-\mu)A(\sigma)-(\lambda-\sigma)A(\mu)]w=0\,, \tag{1.3}$$

solution to the linear eigenproblem

$$A(\sigma)^{-1}A(\mu)w=w\theta, \text{ with } \theta=(\lambda-\mu)/(\lambda-\sigma)\,, \tag{1.4}$$

predicting singularity at

$$\lambda=\mu+\frac{\theta}{1-\theta}(\mu-\sigma)=\frac{1}{1-\theta}\mu-\frac{\theta}{1-\theta}\sigma\,.$$

Let us plot the transformed eigenvalues θ as a function of the original eigenvalues λ in Fig. 1. The extreme eigenvalues θ correspond to the interesting singularities that are close to the pole σ. There will be many eigenvalues close to $\theta=1$ corresponding to λ values far away, while the singularities close to the shift μ will correspond to θ close to zero.

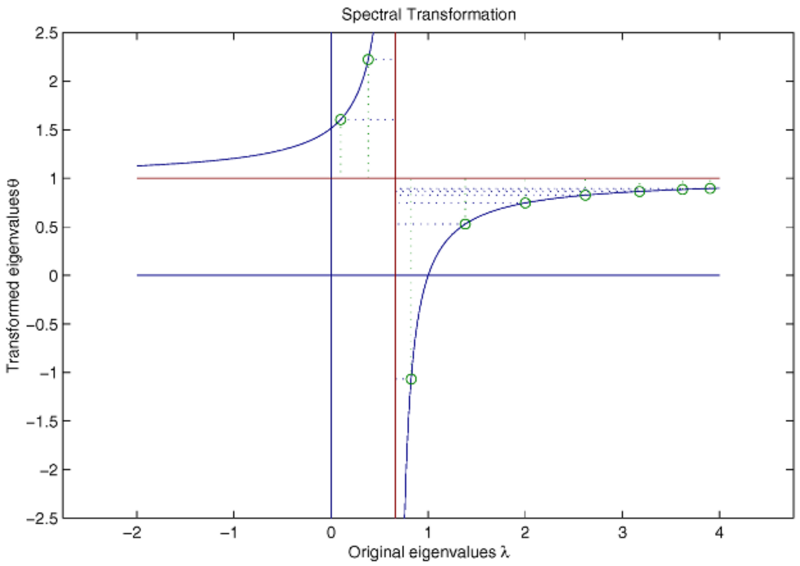


Fig. 1. The Spectral Transformation

Use the Arnoldi algorithm on the generalized eigenvalue problem (1.4), with a Gaussian elimination LU-factorization of $A(\sigma)$ regarded as a preconditioner,

$$A(\sigma)^{-1}A(\mu)V_j=V_jH_{j,j}+R_j\ , \tag{1.5}$$

where V_j is the computed orthogonal basis with j columns, $H_{j,j}$ is a $j\times j$ Hessenberg matrix. The $n\times j$ residual matrix R_j has only its last column filled with a multiple of the next basis vector v_{j+1}.

2 Nonlinear Rational Krylov Algorithm

Our algorithm for the nonlinear problem (1.1) will consist of three parts, one *outer iteration*, where the basis V_j is expanded, one *inner iteration* that computes a new basis vector v_{j+1} for a fixed basis size j, and finally a *lock and purge* phase, where latent vectors x_l are delivered and the basis size is decreased. The inner iteration is run until the residual of the nonlinear problem (1.1) is orthogonal to the basis V_j, while the outer iteration is continued until this residual is small, indicating that one or several latent roots λ_l have converged.

The implementation of our algorithm is closely related to implicitly restarted Arnoldi, IRA [13,2], but the nonlinearity makes the inner iteration necessary. Deflation has to be done precisely when a latent value has converged.

Outer Iteration: ALGORITHM NLRKS, OUTER ITERATION

```
1. Start with pole $\sigma$, shift $\mu$, starting vector $v_1$
2. For $j = 1, 2, \dots$ until *Convergence*,
    (a) Do *inner iteration* to solve projected problem. Update shift $\mu$.
    (b) If $\|r\| < tolc$, *Convergence* break
    (c) $v_{j+1} = r/h_{j+1,j}$, where $h_{j+1,j} = \|r\|/s_j$, (Add vector to basis)
3. Deliver $\lambda_l = \mu$, $x_l = x$, (Latent root and vector)
4. Select new pole $\sigma$ if needed
5. Do *lock and purge*
6. Select new shift $\mu$, (Next latent root).
```

Note that the inner iteration is converged when the residual, $r = A(\sigma)^{-1}A(\mu)$, is orthogonal to the subspace spanned by V_j. If furthermore its norm $\|r\|$ is small enough, we signal convergence of the outer iteration to an eigenvalue of the nonlinear problem (1.1), we call these values *latent values*.

The outer iteration can be regarded as an inverse iteration with shift at the pole σ. If we update the pole, we get a Newton iteration, the same as Rayleigh quotient iteration for the linear eigenvalue problem. However, updating the pole σ would demand refactorization of the matrix, so we keep the same pole σ for a long time, often over several latent values.

As soon as a latent value is converged, we deliver λ_l and its vector x_l as a solution. When a new pole σ is chosen, the latent root λ_l is no longer an eigenvalue of the linearization (1.4).

In the lock and purge operation, we keep the dominating eigenvalues θ of the eigenvalue problem, shifted and inverted at the new pole σ. Some of the eigenvalues that were dominating at a previous pole will become small and can be purged.

Inner Iteration: The task of the *inner* iteration is now to solve the projected problem over the subspace spanned by V_j, and deliver an orthogonal residual r to be added to the basis as v_{j+1}.

ALGORITHM NLRKS, INNER ITERATION

```
1. Start with vector $x = v_j$, shift $\mu$, $s_j = 1$
   $H_{j,j} = \begin{bmatrix} H_{j,j-1} & 0 \end{bmatrix}$, (Hessenberg matrix)
2. Repeat until $\|k_j\|_2 < toln$, (residual orthogonal)
   (a) Compute $r = A(\sigma)^{-1}A(\mu)x$ (Operate)
   (b) Compute $k_j = V_j^* r$, (Gram Schmidt)
   (c) Orthogonalize $r = r - V_j k_j$
   (d) Accumulate $h_j = h_j + k_j s_j^{-1}$
   (e) Compute eigensystem $H_{j,j} S = S \operatorname{diag}(\theta_i)$
   (f) Choose $\theta = \theta_i$ close to 0, $s = s_i$, (Ritz value correction)
   (g) Update $\mu := \mu + \frac{\theta}{1-\theta}(\mu - \sigma) = \frac{1}{1-\theta}\mu - \frac{\theta}{1-\theta}\sigma$
   (h) Update $H_{j,j} := \frac{1}{1-\theta}H_{j,j} - \frac{\theta}{1-\theta}I$.
   (i) Compute $x = V_j s$, (Ritz vector)
```

The updating of the last column of the Hessenberg matrix H in step 2d may need some explanation. First time we reach this step, we fill the column with Gram Schmidt coefficients, precisely as in Arnoldi for a linear eigenproblem (1.5). Next time, we operate on the Ritz vector $x = V_j s$,

$$A(\sigma)^{-1}A(\mu)V_j s = V_j k_j + r$$

Add the recursion from previous iterations to the left of this column vector and get,

$$A(\sigma)^{-1}A(\mu)V_j \begin{bmatrix} I_{j-1} & s_1 \\ 0 & s_j \end{bmatrix} = V_j \begin{bmatrix} H_{j,j-1} & k_j \end{bmatrix} + r e_j^T$$

Postmultiply with the inverse of the elimination matrix and get the updated Hessenberg matrix,

$$\begin{aligned} H'_{j,j} &= \begin{bmatrix} H_{j,j-1} & k_j \end{bmatrix} \begin{bmatrix} I_{j-1} & -s_1 s_j^{-1} \\ 0 & s_j^{-1} \end{bmatrix} \\ &= \begin{bmatrix} H_{j,j-1} & -H_{j,j-1}s_1 s_j^{-1} + k_j s_j^{-1} \end{bmatrix} \\ &= \begin{bmatrix} H_{j,j-1} & h_j + k_j s_j^{-1} \end{bmatrix} \end{aligned}$$

The last equality is a consequence of the updating in step 2h. That update makes sure that s is an eigenvector corresponding to a zero eigenvalue of $H_{j,j}$ which means that $H_{j,j}s = H_{j,j-1}s_1 + h_j s_j = 0$.

The shift μ is updated until close enough to a latent root, and kept as soon as $|\theta| < C|s_j|$. This means that the last column update is a Gram Schmidt orthogonalization and the inner iteration is completed. Note that the last element s_j is not small during early iterations, this is an interesting difference to the linear case, when it gets small as soon as the Krylov space has expanded sufficiently. We need to analyze conditions for convergence of this inner iteration.

When the vector r is delivered as a new basis vector v_{j+1}, it will be orthogonal to the previous basis V_j but it will only be an approximate solution, $\|A(\sigma)^{-1}A(\mu)x\| = \|k_j\| < toln$.

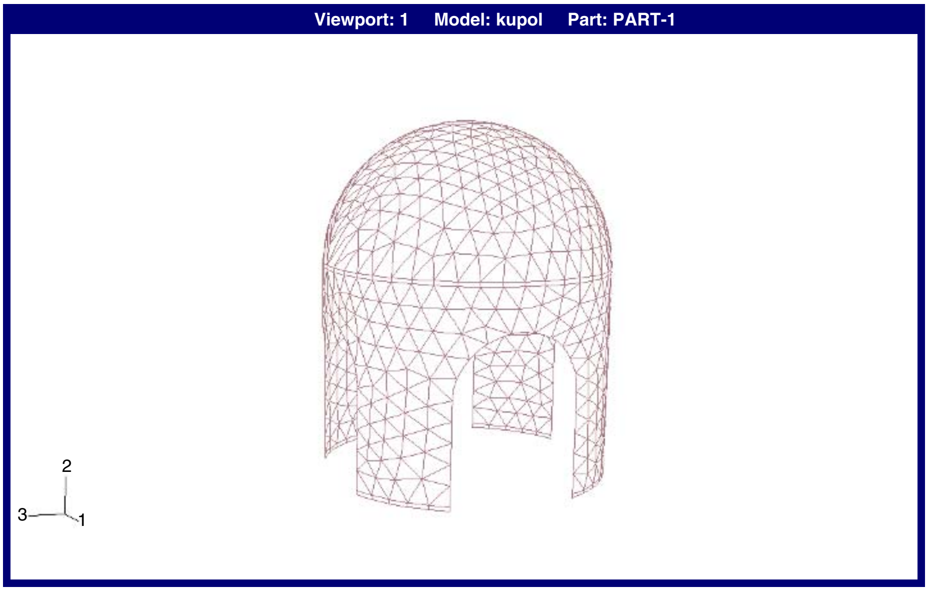


Fig. 2. Finite element model of dome

3 A Numerical Example

We report a run from the thesis of Patrik Hager [5]. He used a FORTRAN90 code on a SGI Origin 2000 computer at Chalmers. The local finite element package FEM 90 was used, together with LAPACK [1] for the dense, and superLU [4] for the sparse matrix operations.

A finite element model of a viscously damped vibrating system is formulated as,

$$A(\lambda) = \lambda^2 M + K - \sum_{m=1}^{\mathrm{NMAT}} \frac{1}{1 + b_m \lambda} \Delta K_m$$

where K, M, ΔK_m denote stiffness, mass and relaxation strength, NMAT is the number of materials and b_m the relaxation constant for the mth material.

We show results for a dome, a stiff half sphere on a stiff tube connected by a damped soft thin layer, this is a model with NMAT $= 3$ materials. A FEM model with shell elements gives matrices of size $n = 13357$, see Fig. 2!

We seek the 50 smallest latent roots in the third quadrant, see the plot in Fig. 3! Note that this figure is not equally scaled, actually all latent roots are very close to the imaginary axis.

The algorithm is restricted to run up to 30 basis vectors. Follow the progress in Fig. 4! In average 2.15 inner iterations were needed in each RKS step and 15.2 RKS steps for each latent root. Note that the shift μ is updated frequently, while the same pole σ is used for several iterations.

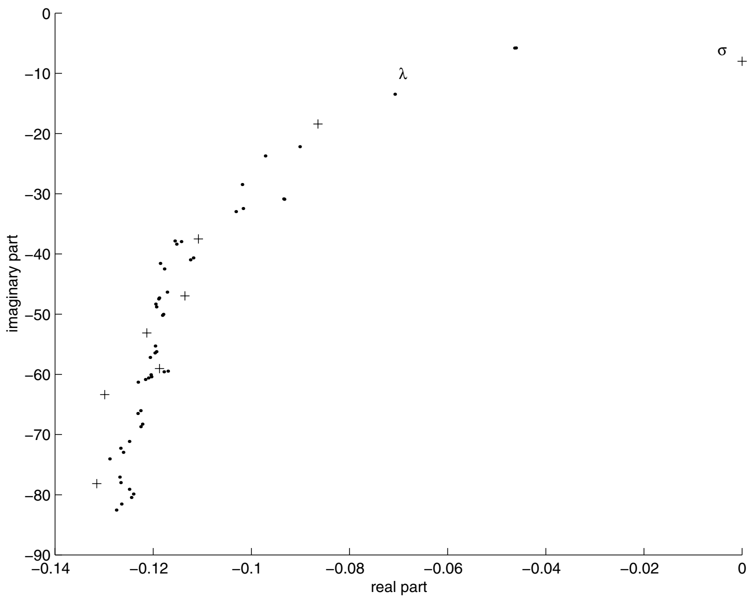


Fig. 3. Latent roots of dome. Dots are latent roots, Pluses are poles σ used. Note unequal scaling!

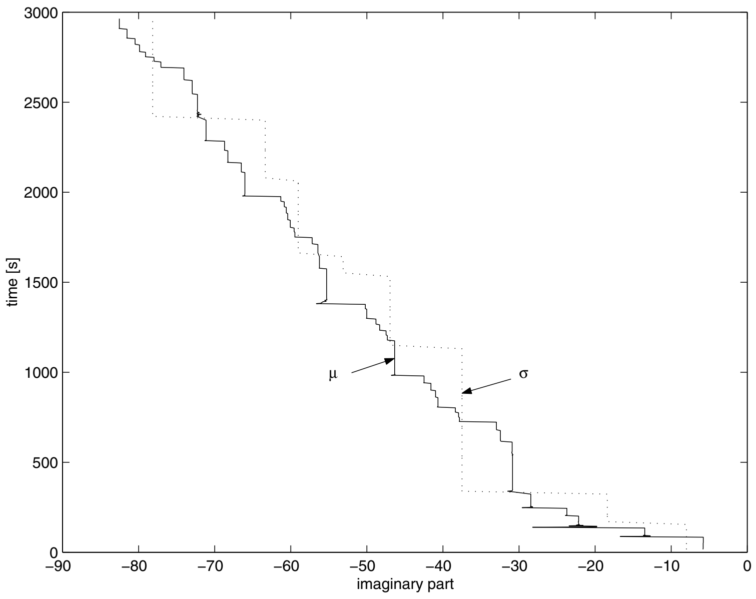


Fig. 4. Follow algorithm for dome. Time flows upwards, imaginary parts of shift μ and pole σ on x-axis

References

1. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, second ed., 1995.
2. Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., *Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide*, SIAM, Philadelphia, 2000.
3. T. Betcke and H. Voss, *A Jacobi–Davidson–type projection method for nonlinear eigenvalue problems*, Future Generation Computer Systems, 20 (2004), pp. 363 – 372.
4. J. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM J. Matr. Anal. Appl., 20 (1999), pp. 720–755.
5. P. Hager, *Eigenfrequency Analysis - FE-Adaptivity and a Nonlinear Eigenproblem Algorithm*, PhD thesis, Chalmers University of Technology, Göteborg, Sweden, 2001.
6. E. Jarlebring, *Krylov methods for nonlinear eigenvalue problems*, Master's thesis, NADA KTH, Stockholm Sweden, 2003. TRITA-NA-E03042.
7. V. N. Kublanovskaja, *On an applicaton of Newton's method to the determination of eigenvalues of λ-matrices*, Dokl. Akad. Nauk SSSR, 188 (1969), pp. 1240–1241. Page numbers may refer to AMS translation series.
8. K. H. A. Olsson, Model Order Reduction in FEMLAB by Dual Rational Arnoldi. Licentiate thesis, Chalmers University of Technology, Göteborg, Sweden, 2002.
9. A. Ruhe, *Algorithms for the nonlinear algebraic eigenvalue problem*, SIAM J. Num. Anal, 10 (1973), pp. 674–689.
10. A. Ruhe, *Rational Krylov, a practical algorithm for large sparse nonsymmetric matrix pencils*, SIAM J. Sci. Comp., 19 (1998), pp. 1535–1551.
11. A. Ruhe, *A Rational Krylov algorithm for nonlinear matrix eigenvalue problems*, vol. 268 of Zapiski Nauchnyh Seminarov POMI, S:t Petersburg, 2000, pp. 176–180.
12. A. Ruhe and D. Skoogh, *Rational Krylov algorithms for eigenvalue computation and model reduction*, in Applied Parallel Computing. Large Scale Scientific and Industrial Problems., B. Kågström, J. Dongarra, E. Elmroth, and J. Waśniewski, eds., Lecture Notes in Computer Science, No. 1541, 1998, pp. 491–502.
13. D. C. Sorensen, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J. Matr. Anal. Appl., 13 (1992), pp. 357–385.
14. H. Voss, *An Arnoldi method for nonlinear eigenvalue problems*, BIT Numerical Mathematics, 44 (2004), pp. 387 – 401.
15. H. Voss, *A Jacobi–Davidson method for nonlinear eigenproblems*, in Computational Science – ICCS 2004, 4th International Conference, Kraków, Poland, June 6–9 2004, Proceedings, Part II, M. Buback, G. van Albada, P. Sloot, and J. Dongarra, eds., vol. 3037 of Lecture Notes in Computer Science, Berlin, Heidelberg, New York, 2004, Springer Verlag, pp. 34–41.

Algebraic Sub-structuring for Electromagnetic Applications

Chao Yang[1], Weiguo Gao[1], Zhaojun Bai[2], Xiaoye S. Li[1], Lie-Quan Lee[3], Parry Husbands[1], and Esmond G. Ng[1]

[1] Computational Research Division
Lawrence Berkeley National Lab
Berkeley CA 94720, USA
{CYang,WGGao,XSLi,PJRHusbands,EGNg}@lbl.gov
[2] Department of Computer Science
The University of California at Davis
Davis, CA 95616, USA
bai@cs.ucdavis.edu
[3] Stanford Linear Accelerator Center
Menlo Park, CA 94025, USA
liequan@slac.stanford.edu

Abstract. Algebraic sub-structuring refers to the process of applying matrix reordering and partitioning algorithms to divide a large sparse matrix into smaller submatrices from which a subset of spectral components are extracted and combined to form approximate solutions to the original problem. In this paper, we show that algebraic sub-structuring can be effectively used to solve generalized eigenvalue problems arising from the finite element analysis of an accelerator structure.

1 Introduction

Sub-structuring is a commonly used technique for studying the static and dynamic properties of large engineering structures [3,6,11]. The basic idea of sub-structuring is analogous to the concept of domain-decomposition widely used in the numerical solution of partial differential equations [13]. By dividing a large structure model or computational domain into a few smaller components (sub-structures), one can often obtain an approximate solution to the original problem from a linear combination of solutions to similar problems defined on the sub-structures. Because solving problems on each sub-structure requires far less computational power than what would be required to solve the entire problem as a whole, sub-structuring can lead to a significant reduction in the computational time required to carry out a large-scale simulation and analysis.

The automated multi-level sub-structuring (AMLS) method introduced in [1,7] is an extension of a simple sub-structuring method called *component mode synthesis* (CMS) [3,6] originally developed in the 1960s to solve large-scale eigenvalue problems. The method has been used successfully in the vibration and acoustic analysis of large-scale finite element models of automobile bodies [7,9]. The timing results reported in [7,9] indicate that AMLS is significantly faster than conventional Lanczos-based approaches

[10,5]. However, it is important to note that the accuracy achieved by a sub-structuring method such as AMLS is typically lower than that achieved by the standard Lanczos algorithm. The method is most valuable when a large number of eigenpairs with relatively low accuracy are of interest.

In [15], we examined sub-structuring methods for solving large-scale eigenvalue problems from a purely algebraic point of view. We used the term *algebraic sub-structuring* to refer to the process of applying matrix reordering and partitioning algorithms (such as the *nested dissection* algorithm [4]) to divide a large sparse matrix into smaller submatrices from which a subset of spectral components are extracted and combined to form an approximate solution to the original eigenvalue problem. Through an algebraic manipulation, we identified the critical conditions under which algebraic sub-structuring works well. In particular, we observed an interesting connection between the accuracy of an approximate eigenpair obtained through sub-structuring and the distribution of components of eigenvectors associated with a canonical matrix pencil congruent to the original problem. We developed an error estimate for the approximation to the smallest eigenpair, which we will summarize in this paper. The estimate leads to a simple heuristic for choosing spectral components from each sub-structure.

Our interest in algebraic sub-structuring is motivated in part by an application arising from the simulation of the electromagnetic field associated with next generation particle accelerator design [8]. We show in this paper that algebraic sub-structuring can be used effectively to compute the cavity resonance frequencies and the electromagnetic field generated by a linear particle accelerator model.

Throughout this paper, capital and lower case Latin letters denote matrices and vectors respectively, while lower case Greek letters denote scalars. An $n \times n$ identity matrix will be denoted by I_n. The j-th column of the identity matrix is denoted by e_j. The transpose of a matrix A is denoted by A^T. We use $\|x\|$ to denote the standard 2-norm of x, and use $\|x\|_M$ to denote the M-norm defined by $\|x\|_M = \sqrt{x^T M x}$. We will use $\angle_M(x, y)$ to denote the M-inner product induced acute angle (M-angle for short) between x and y. This angle can be computed from $\cos \angle_M(x, y) = x^T M y / \|x\|_M \|y\|_M$. A matrix pencil (K, M) is said to be *congruent* to another pencil (A, B) if there exits a nonsingular matrix P, such that $A = P^T K P$ and $B = P^T M P$.

2 Algebraic Sub-structuring

In this section, we briefly describe a single-level algebraic sub-structuring algorithm. This is also known as the *component synthesis method* (CMS) in the engineering literature [6]. Our description does not make use of any information regarding the geometry or the physical structure on which the original problem is defined.

We are concerned with solving the following generalized algebraic eigenvalue problem

$$Kx = \lambda M x, \tag{2.1}$$

where K is symmetric and M is symmetric positive definite. We assume K and M are both sparse. They may or may not have the same sparsity pattern. Suppose the rows and columns of K and M have been permuted so that these matrices can be partitioned as

$$K = \begin{array}{c|ccc} & n_1 & n_2 & n_3 \\ \hline n_1 & K_{11} & & K_{13} \\ n_2 & & K_{22} & K_{23} \\ n_3 & K_{13}^T & K_{23}^T & K_{33} \end{array} \quad \text{and} \quad M = \begin{array}{c|ccc} & n_1 & n_2 & n_3 \\ \hline n_1 & M_{11} & & M_{13} \\ n_2 & & M_{22} & M_{23} \\ n_3 & M_{13}^T & M_{23}^T & M_{33} \end{array}, \tag{2.2}$$

where the labels n_1, n_2 and n_3 denote the dimensions of each sub-matrix block. The permutation can be accomplished by applying a matrix ordering and partitioning algorithm such as the nested dissection algorithm [4] to the matrix $K + M$.

The pencils (K_{11}, M_{11}) and (K_{22}, M_{22}) now define two algebraic sub-structures that are connected by the third block rows and columns of K and M which we will refer to as the *interface* block. We assume that n_3 is much smaller than n_1 and n_2.

A single-level algebraic sub-structuring algorithm proceeds by performing a block factorization

$$K = LDL^T, \tag{2.3}$$

where

$$L = \begin{pmatrix} I_{n_1} & & \\ & I_{n_2} & \\ K_{13}^T K_{11}^{-1} & K_{23}^T K_{22}^{-1} & I_{n_3} \end{pmatrix} \quad \text{and} \quad D = \begin{pmatrix} K_{11} & & \\ & K_{22} & \\ & & \widehat{K}_{33} \end{pmatrix}.$$

The last diagonal block of D, often known as the *Schur complement*, is defined by

$$\widehat{K}_{33} = K_{33} - K_{13}^T K_{11}^{-1} K_{13} - K_{23}^T K_{22}^{-1} K_{23}.$$

The inverse of the lower triangular factor L defines a congruence transformation that, when applied to the matrix pencil (K, M), yields a new matrix pencil $(\widehat{K}, \widehat{M})$:

$$\widehat{K} = L^{-1} K L^{-T} = D \quad \text{and} \quad \widehat{M} = L^{-1} M L^{-T} = \begin{pmatrix} M_{11} & & \widehat{M}_{13} \\ & M_{22} & \widehat{M}_{23} \\ \widehat{M}_{13}^T & \widehat{M}_{23}^T & \widehat{M}_{33} \end{pmatrix}. \tag{2.4}$$

The pencil $(\widehat{K}, \widehat{M})$ is often known as the *Craig-Bampton* form [3] in structural engineering. Note that the eigenvalues of $(\widehat{K}, \widehat{M})$ are identical to those of (K, M), and the corresponding eigenvectors $\widehat{x}$ are related to the eigenvectors of the original problem (2.1) through $\widehat{x} = L^T x$.

The sub-structuring algorithm constructs a subspace spanned by

$$S = \begin{array}{c|ccc} & k_1 & k_2 & n_3 \\ \hline n_1 & S_1 & & \\ n_2 & & S_2 & \\ n_3 & & & I_{n_3} \end{array} \tag{2.5}$$

where S_1 and S_2 consist of k_1 and k_2 selected eigenvectors of (K_{11}, M_{11}) and (K_{22}, M_{22}) respectively. These eigenvectors will be referred to as *sub-structure modes* in the discussion that follows. Note that k_1 and k_2 are typically much smaller than n_1 and n_2, respectively.

The approximation to the desired eigenvalues and eigenvectors of the pencil $(\widehat{K}, \widehat{M})$ are obtained by projecting the pencil $(\widehat{K}, \widehat{M})$ onto the subspace spanned by S, i.e., we seek θ and $q \in \mathbb{R}^{k_1+k_2+n_3}$ such that

$$(S^T \widehat{K} S)q = \theta(S^T \widehat{M} S)q. \tag{2.6}$$

It follows from the standard Rayleigh-Ritz theory [12, page 213] that θ serves as an approximation to an eigenvalue of (K, M), and the vector formed by $z = L^{-T} Sq$ is the approximation to the corresponding eigenvector.

One key aspect of the algebraic sub-structuring algorithm is that k_i can be chosen to be much smaller than n_i. Thus, S_i can be computed by a shift-invert Lanczos procedure. The cost of this computation is generally small compared to the rest of the computation, especially when this algorithm is extended to a multi-level scheme. Similarly, because n_3 is typically much smaller than n_1 and n_2, the dimension of the projected problem (2.6) is significantly smaller than that of the original problem. Thus, the cost of solving (2.6) is also relatively small.

Decisions must be made on how to select eigenvectors from each sub-structure. The selection should be made in such a way that the subspace spanned by the columns of S retains a sufficient amount of spectral information from (K, M). The process of choosing appropriate eigenvectors from each sub-structure is referred to as *mode selection* [15].

The algebraic sub-structuring algorithm presented here can be extended in two ways. First, the matrix reordering and partitioning scheme used to create the block structure of (2.2) can be applied recursively to (K_{11}, M_{11}) and (K_{22}, M_{22}) respectively to produce a multi-level division of (K, M) into smaller sub-matrices. The reduced computational cost associated with finding selected eigenpairs from these even smaller sub-matrices further improves the efficiency of the algorithm. Second, one may replace I_{n_3} in (2.5) with a subset of eigenvectors of the interface pencil $(\widehat{K}_{33}, \widehat{M}_{33})$. This modification will further reduce the computational cost associated with solving the projected eigenvalue problem (2.6). A combination of these two extensions yields the AMLS algorithm presented in [7]. However, we will limit the scope of our presentation to a single level sub-structuring algorithm in this paper.

3 Accuracy and Error Estimation

One of the natural questions one may ask is how much accuracy we can expect from the approximate eigenpairs obtained through algebraic sub-structuring. The answer to this question would certainly depend on how S_1 and S_2 are constructed in (2.5). This issue is carefully examined in [15]. In this section, we will summarize the error estimate results established in [15].

To simplify the discussion, we will work with the matrix pencil $(\widehat{K}, \widehat{M})$, where $\widehat{K}$ and $\widehat{M}$ are defined in (2.4). As we noted earlier, $(\widehat{K}, \widehat{M})$ and (K, M) have the same set of eigenvalues. If $\widehat{x}$ is an eigenvector of $(\widehat{K}, \widehat{M})$, then $x = L^{-T}\widehat{x}$ is an eigenvector of (K, M), where L is the transformation defined in (2.3).

If $(\mu_j^{(i)}, v_j^{(i)})$ is the j-th eigenpair of the i-th sub-problem, i.e.,

$$K_{ii} v_j^{(i)} = \mu_j^{(i)} M_{ii} v_j^{(i)},$$

where $(v_j^{(i)})^T M_{ii} v_k^{(i)} = \delta_{j,k}$, and $\mu_j^{(i)}$ has been ordered such that

$$\mu_1^{(i)} \le \mu_2^{(i)} \le \cdots \le \mu_{n_i}^{(i)}, \tag{3.7}$$

then we can express $\widehat{x}$ as

$$\widehat{x} = \begin{pmatrix} V_1 & & \\ & V_2 & \\ & & I_{n_3} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}, \tag{3.8}$$

where $V_i = (v_1^{(i)}\ v_2^{(i)}\ ...\ v_{n_i}^{(i)})$, and $y = (y_1^T, y_2^T, y_3^T)^T \neq 0$.

It is easy to verify that y satisfies the following *canonical* generalized eigenvalue problem

$$\begin{pmatrix} \Sigma_1 & & \\ & \Sigma_2 & \\ & & \widehat{K}_{33} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \lambda \begin{pmatrix} I_{n_1} & & G_{13} \\ & I_{n_2} & G_{23} \\ G_{13}^T & G_{23}^T & \widehat{M}_{33} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}, \tag{3.9}$$

where $\Sigma_i = \text{diag}(\mu_1^{(i)}, \mu_2^{(i)}, \ldots, \mu_{n_i}^{(i)})$, $G_{i3} = V_i^T \widehat{M}_{i3}$ for $i = 1, 2$. This pencil is clearly congruent to the pencils $(\widehat{K}, \widehat{M})$ and (K, M). Thus it shares the same set of eigenvalues with that of (K, M).

If $\widehat{x}$ can be well approximated by a linear combination of the columns of S, as suggested by the description of the the algorithm in Section 2, then the vector y_i $(i = 1, 2)$ must contain only a few large entries. All other components of y_i are likely to be small and negligible.

In [15], we showed that

$$|y_i| = \text{diag}\left(\rho_\lambda(\mu_1^{(i)}), \rho_\lambda(\mu_2^{(i)}), \cdots, \rho_\lambda(\mu_{n_i}^{(i)}) \right) g^{(i)}, \tag{3.10}$$

where $g^{(i)} = |e_j^T G_{i3} y_3|$, and

$$\rho_\lambda(\omega) = |\lambda/(\omega - \lambda)|. \tag{3.11}$$

When elements of $g^{(i)}$ can be bounded (from above and below) by a moderate constant, the magnitude of $|e_j^T y_i|$ is essentially determined by $\rho_\lambda(\mu_j^{(i)})$ which is called a *ρ-factor* in [15].

It is easy to see that $\rho_\lambda(\mu_j^{(i)})$ is large when $\mu_j^{(i)}$ is close to λ, and it is small when $\mu_j^{(i)}$ is away from λ. For the smallest eigenvalue (λ_1) of (K, M), it is easy to show that $\rho_{\lambda_1}(\mu_j^{(i)})$ is monotonically decreasing with respect to j. Thus, if λ_1 is the desired eigenvalue, one would naturally choose the matrix S_i in (2.5) to contain only the leading k_i columns of V_i, for some $k_i \ll n_i$.

If we define h_i by

$$e_j^T h_i = \begin{cases} 0 & \text{for } j \le k_i, \\ e_j^T y_i & \text{for } k_i < j \le n_i, \end{cases} \tag{3.12}$$

then following theorem, which we proved in [15], provides an *a priori* error estimate for the Rayleigh-Ritz approximation to $(\lambda_1, \widehat{x}_1)$ from the subspace spanned by columns of S defined in (2.5).

Theorem 1. *Let $\widehat{K}$ and $\widehat{M}$ be the matrices defined in (2.4). Let $(\lambda_i, \widehat{x}_i)$ $(i = 1, 2, ...n)$ be eigenpairs of the pencil $(\widehat{K}, \widehat{M})$, ordered so that $\lambda_1 < \lambda_2 \le \cdots \le \lambda_n$. Let (θ_1, u_1) be the Rayleigh-Ritz approximation to $(\lambda_1, \widehat{x}_1)$ from the space spanned by the columns of S defined in (2.5). Then*

$$\theta_1 - \lambda_1 \le (\lambda_n - \lambda_1)(h_1^T h_1 + h_2^T h_2), \tag{3.13}$$

$$\sin \angle_{\widehat{M}}(u_1, \hat{x}_1) \le \sqrt{\frac{\lambda_n - \lambda_1}{\lambda_2 - \lambda_1}} \sqrt{h_1^T h_1 + h_2^T h_2}, \tag{3.14}$$

where h_i $(i = 1, 2)$ is defined by (3.12).

Theorem 1 indicates that the accuracy of (θ_1, u_1) is proportional to the size of $h_1^T h_1 + h_2^T h_2$, a quantity that provides a cumulative measure of the “truncated" components in (3.8).

If $\rho_{\lambda_1}(\mu_j^{(i)}) < \tau < 1$ holds for $k_i < j \le n_i$, and if $e_j^T g^{(i)} \le \gamma$ for some moderate sized constant γ, we can show [15] that $h_1^T h_1 + h_2^T h_2$ can be bounded by a quantity that is independent of the number of non-zero elements in h_i. Consequently, we can establish the following bounds:

$$\frac{\theta_1 - \lambda_1}{\lambda_1} \le (\lambda_n - \lambda_1)(2\alpha\tau), \tag{3.15}$$

$$\sin \angle_{\widehat{M}}(\hat{x}_1, u_1) \le \sqrt{\lambda_1 \left(\frac{\lambda_n - \lambda_1}{\lambda_2 - \lambda_1} \right)} \sqrt{2\alpha\tau}, \tag{3.16}$$

where $\alpha = \gamma^2/\delta$.

We should mention that (3.15) and (3.16) merely provide a qualitative estimate of the error in the Ritz pair (θ_1, u_1) in terms of the threshold τ that may be used as a heuristic in practice to determine which spectral components of a sub-structure should be included in the subspace S defined in (2.5). It is clear from these inequalities that a smaller τ, which typically corresponds to a selection of more spectral components from each sub-structure, leads to a more accurate Ritz pair (θ_1, u_1).

4 Numerical Experiment

We show by an example that algebraic sub-structuring can be used to compute approximate cavity resonance frequencies and the electromagnetic field associated with a small

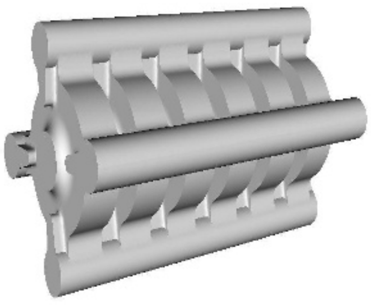


Fig. 1. The finite element model corresponding to a 6-cell damped detuned structure

accelerator structure. The matrix pencil used in this example is obtained from a finite element model of a six-cell Damped Detuned accelerating Structure (DDS) [8]. The three dimensional geometry of the model is shown in Figure 1. The dimension of the pencil (K, M) is $n = 5584$. The stiffness matrix K has 580 zero rows and columns. These zero rows and columns are produced by a particular hierarchical vector finite element discretization scheme [14]. Because K is singular, we cannot perform the block elimination in (2.3) directly. A deflation scheme is developed in [15] to overcome this difficulty. The key idea of the deflation scheme is to replace K_{ii}^{-1} $(i = 1, 2)$ with a pseudo-inverse in the congruence transformation calculation. We refer the reader to [15] for the algorithmic details. To facilitate deflation, we perform a two-stage matrix reordering described in [15]. Figure 2 shows the non-zero patterns of the permuted K and M.

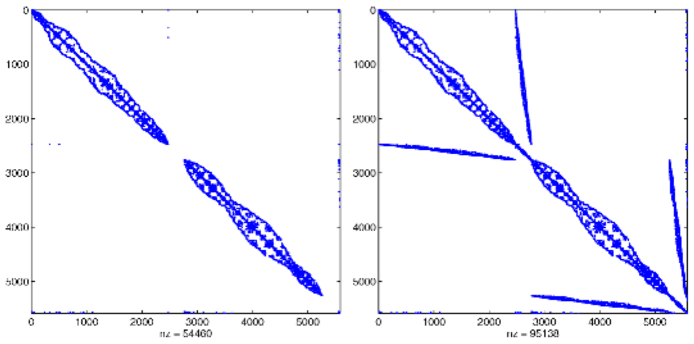


Fig. 2. The non-zero pattern of the permuted stiffness matrix K (left) and the mass matrix M (right) associated with the 6-cell DDS model

We plot the approximate ρ-factors associated with smallest eigenvalue of the deflated problem in Figure 3. The approximation is made by replacing λ_1 (which we do not know in advance) in (3.11) with $\sigma \equiv \min(\mu_1^{(1)}, \mu_1^{(2)})/2$. We showed in [15] that such an approximation does not alter the qualitative behavior of the ρ-factor. Three different

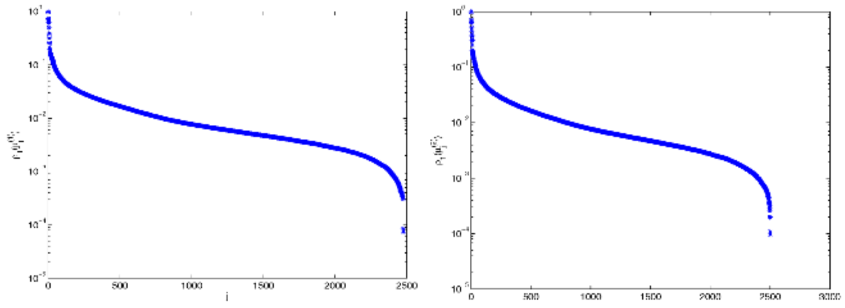


Fig. 3. The approximate ρ-factors associated with each sub-structure of the 6-cell DDS model

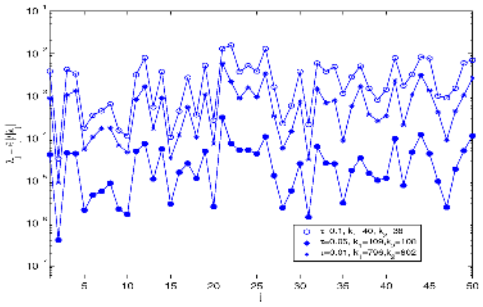


Fig. 4. The relative error of the smallest 50 Ritz values extracted from three subspaces constructed by using different choices of the ρ-factor thresholds (τ values) for the DDS model

choices of τ values were used as the ρ-factor thresholds ($\tau = 0.1, 0.05, 0.01$) for selecting sub-structure modes, i.e., we only select sub-structure modes that satisfy $\rho_\sigma(\mu_j^{(i)}) \geq \tau$.

The relative accuracy of the 50 smallest non-zero Ritz values extracted from the subspaces constructed with these choices of τ values is displayed in Figure 4.

We observe that with $\tau = 0.1$, θ_1 has roughly three digits of accuracy, which is quite sufficient for this particular discretized model. If we decrease τ down to 0.01, most of the smallest 50 non-zero Ritz values have at least 4 to 5 digits of accuracy.

The least upper bound for elements of $g^{(i)}$ used in (3.10) is $\gamma = 0.02$. Thus the ρ-factor gives an over-estimate of elements of $|y_i|$ in this case. In Figure 5, we plot elements of $|y_1|$ and $|y_2|$, where $(y_1^T, y_2^T, y_3^T)^T$ is the eigenvector associated with the smallest non-zero eigenvalue of (3.9). For simplicity, we excluded the elements of $|y_1|$ and $|y_2|$ corresponding to the null space of (K_{11}, M_{11}) and (K_{22}, M_{22}), which have been deflated from our calculations. We observe that $|e_j^T y_i|$ is much smaller than $\rho_\sigma(\mu_j^{(i)})$, and it decays much faster than the the ρ-factor also.

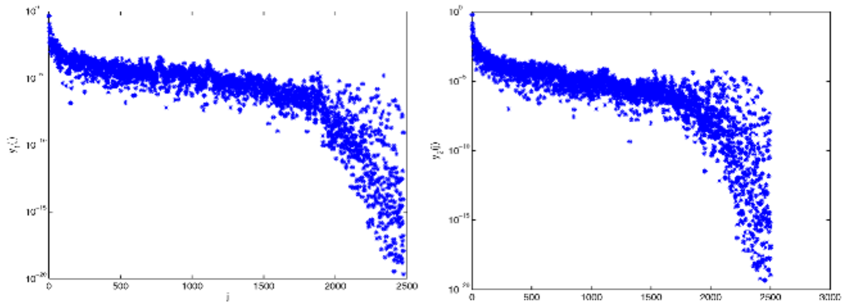


Fig. 5. The magnitude of $|y_1|$ (left) and $|y_2|$ (right) elements, where $(y_1^T, y_2^T, y_3^T)^T$ is the eigenvector corresponding to the smallest eigenvalue of the canonical problem (3.9) associated with the DDS model

5 Concluding Remarks

In this paper, we discussed the possibility of using algebraic sub-structuring to solve large-scale eigenvalue problems arising from electromagnetic simulation. We examined the accuracy of the method based on the analysis developed in [15]. A numerical example is provided to demonstrate the effectiveness of the method.

We should point out that the block elimination and congruence transformation performed in algebraic sub-structuring can be costly in terms of memory usage. However, since no triangular solves on the full matrix, which are typically used in a standard shift-invert Lanczos algorithm, are required, an efficient multi-level out-of-core implementation is possible. We will discuss the implementation issues and comparison with other methods in a future study.

References

1. J. K. Bennighof. Adaptive multi-level substructuring method for acoustic radiation and scattering from complex structures. In A. J. Kalinowski, editor, *Computational methods for Fluid/Structure Interaction*, volume 178, pages 25–38, AMSE, New York, November, 1993.
2. F. Bourquin. Component mode synthesis and eigenvalues of second order operators: Discretization and algorithm. *Mathematical Modeling and Numerical Analysis*, 26:385–423, 1992.
3. R. R. Craig and M. C. C. Bampton. Coupling of substructures for dynamic analysis. *AIAA Journal*, 6:1313–1319, 1968.
4. A. George. Nested dissection of a regular finite element mesh. *SIAM J. Num. Anal.*, 10:345–363, 1973.
5. R. G. Grimes, J. G. Lewis, and H. D. Simon. A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems. *SIAM Journal on Matrix Analysis and Applications*, 15(1):228–272, January 1994.
6. W. C. Hurty. Vibrations of structure systems by component-mode synthesis. *Journal of the Engineering Mechanics Dvision, ASCE*, 86:51–69, 1960.

7. M. F. Kaplan. *Implementation of Automated Multilevel Substructuring for Frequency Response Analysis of Structures*. PhD thesis, University of Texas at Austin, Austin, TX, December 2001.
8. K. Ko, N. Folwell, L. Ge, A. Guetz, V. Ivanov, L. Lee, Z. Li, I. Malik, W. Mi, C. Ng, and M. Wolf. Electromagnetic systems simulation - from simulation to fabrication. SciDAC report, Stanford Linear Accelerator Center, Menlo Park, CA, 2003.
9. A. Kropp and D. Heiserer. Efficient broadband vibro-accoutic analysis of passenger car bodies using an FE-based component mode synthesis approach. In *Proceedings of the fifth World Congress on Computational Mechanics (WCCM V)*, Vienna University of Technology, 2002.
10. R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide – Solution of Large-scale eigenvalue problems with implicitly restarted Arnoldi Methods*. SIAM, Philadelphia, PA., 1999.
11. R. H. MacNeal. Vibrations of composite systems. Technical Report OSRTN-55-120, Office of Scientific Research, Air Research of Scientific Research and Development Command, 1954.
12. B. N. Parlett. *The Symmetric Eigenvalue Problem*. Prentice-Hall, 1980.
13. B. Smith, P. Bjørstad, and W. Gropp. *Domain Decomposition*. Cambridge University Press, Cambridge, UK., 1996.
14. Din-Kow Sun, Jin-Fa Lee, and Zoltan Cendes. Construction of nearly orthogonal Nedelec bases for rapid convergence with multilevel preconditioned solvers. *SIAM Journal on Scientific Computing*, 23(4):1053–1076, 2001.
15. C. Yang, W. Gao, Z. Bai, X. Li, L. Lee, P. Husbands, and E. G. Ng. An Algebraic Sub-structuring Algorithm for Large-scale Eigenvalue Calculation. *Technical Report, LBNL-55055*, Submitted to SIAM Journal on Scientific Computing.

Parallel Processing in Science and Engineering: An Introduction

Organizer: Adam W. Bojańczyk

School of Electrical and Computer Engineering
Cornell University
Ithaca, NY, 14850, USA
adamb@ece.cornell.edu

Introduction

This minisymposium addressed selected aspects of parallel and distributing computing as they arise in engineering, industrial and scientific computing. Both non-traditional applications as well as relevant software tools were presented.

A big concern of HPC is the development of software that optimizes the performance of a given computer. Several papers addressed this problem.

Several papers discussed systems for automatic generation of provably correct linear algebra solvers. These systems take as input information about the operation to be performed and about the target architectures, and generate an optimized library routine for that operation even if the operation has not previously been implemented. These system speed up creation of application specific libraries for the target architectures and target applications.

Another pressing problem is that of load balancing on SMP clusters which run hybrid MPI and OpenMP applications. In a paper addressing this problem a software system is presented which automatically decreases or increase number of threads on a node depending on the dynamically changing workload of the node relatively to other nodes. This feature helps keeping workloads on all nodes balanced and hence increases the utilization of the cluster.

The high cost of HPC systems can keep them beyond the reach of many potential users. A solution to this problem is offered by grid computing where remote] computing facilities are made available to users via internet. In the grid paradigm all small computations are performed on a single local computer, while large-scale computations are automatically distributed to more powerful computing resource on the grid. In this scenario there is a need for interfacing software libraries on remote resources. This problem is address in a paper which presents a prototype for semi-automatic generation of NetSolve interfaces for complete numerical software libraries. The prototype is demonstrated by generating NetSolve interfaces for the complete SLICOT library for the design and analysis of control systems.

The ongoing development of advanced computers provides the potential for solving increasingly difficult computational problems either in terms of massive datasets, real time requirements or computational complexity.

One promising application area of HPC is in risk management and financial engineering. Commodity clusters , with point-and-click access from a desktop, offer substantial computing power for the financial analyst or risk manager to consider complicated financial models that arise in computational risk management. A paper on financial risk management demonstrates the convenience and power of cluster approaches when working with pricing and hedging of large portfolios, Value-at-Risk, credit risk computations, or sensitivity analysis.

Another computation intensive application where HPC are indispensable is that of Space-Time Adaptive Processing (STAP). STAP refers to adaptive radar processing algorithms which operate on data collected from both multiple sensors and multiple pulses to cancel interference and detect signal sources. As the optimal method cannot be implemented in real-time, many different STAP heuristic which trade computational complexity with accuracy of the detection have been proposed. The last paper in the minisymposium introduces a software framework called ALPS for rapid prototyping and optimizing these various heuristic methods. Within this framework, users describe STAP heuristics with basic building blocks. The software finds the optimal parallel implementation of the heuristic as well as assesses its detection properties.

Rapid Development of High-Performance Linear Algebra Libraries

Paolo Bientinesi[1], John A. Gunnels[4], Fred G. Gustavson[4], Greg M. Henry[3], Margaret Myers[1], Enrique S. Quintana-Ortí[2], and Robert A. van de Geijn[1]

[1] Department of Computer Sciences
The University of Texas at Austin
{pauldj,myers,rvdg}@cs.utexas.edu
[2] Universidad Jaume I, Spain
quintana@inf.uji.es
[3] Intel Corp.
greg.henry@intel.com
[4] IBM's T.J. Watson Research Center
{gunnels,fg2}@us.ibm.com

Abstract. We present a systematic methodology for deriving and implementing linear algebra libraries. It is quite common that an application requires a library of routines for the computation of linear algebra operations that are not (exactly) supported by commonly used libraries like LAPACK. In this situation, the application developer has the option of casting the operation into one supported by an existing library, often at the expense of performance, or implementing a custom library, often requiring considerable effort. Our recent discovery of a methodology based on formal derivation of algorithm allows such a user to quickly derive proven correct algorithms. Furthermore it provides an API that allows the so-derived algorithms to be quickly translated into high-performance implementations.

1 Introduction

We have recently written a series of journal papers where we illustrate to the HPC community the benefits of the formal derivation of algorithms [7,2,11,3]. In those papers, we show that the methodology greatly simplifies the derivation and implementation of algorithms for a broad spectrum of dense linear algebra operations. Specifically, it has been successfully applied to all Basic Linear Algebra Subprograms (BLAS) [9,5,4], most operations supported by the Linear Algebra Package (LAPACK) [1], and many operations encountered in control theory supported by the RECSY library [8]. We illustrate the methodology and its benefits by applying it to the inversion of a triangular matrix, $L := L^{-1}$, an operation supported by the LAPACK routine DTRTRI.

2 A Worksheet for Deriving Linear Algebra Algorithms

In Fig. 1, we give a generic "worksheet" for deriving a large class of linear algebra algorithms. Expressions in curly-brackets (Steps 1a, 1b, 2, 2,3, 6, 7) denote predicates

Step	Annotated Algorithm: $[D, E, F, \ldots] = \mathrm{op}(A, B, C, D, \ldots)$
1a	$\{P_{\mathrm{pre}}\}$
4	**Partition** **where**
2	$\{P_{\mathrm{inv}}\}$
3	**while** G **do**
2,3	$\{(P_{\mathrm{inv}}) \wedge (G)\}$
5a	**Repartition** **where**
6	$\{P_{\mathrm{before}}\}$
8	S_U
7	$\{P_{\mathrm{after}}\}$
5b	**Continue with**
2	$\{P_{\mathrm{inv}}\}$
	enddo
2,3	$\{(P_{\mathrm{inv}}) \wedge \neg (G)\}$
1b	$\{P_{\mathrm{post}}\}$

Fig. 1. Worksheet for developing linear algebra algorithms

that describe the state of the various variables at the given point of the algorithm. The statements between the predicates (Steps 3, 4, 5a, 5b, 8) are chosen in such a way that, at the indicated points in the algorithm, those predicates hold. In the left column of Fig. 1, the numbering of the steps reflects the order in which the items are filled in.

3 Example: Triangular Matrix Inversion

Let us consider the example $L := L^{-1}$ where L is an $m \times m$ lower triangular matrix. This is similar to the operation provided by the LAPACK routine DTRTRI [4]. In the discussion below the "Steps" refer to the step numbers in the left column of Figs. 1 and 2.

Step 1: Determine P_{pre} and P_{post}. The conditions before the operation commences (the precondition) can be described by the predicate indicated in Step 1a in Fig. 2. Here $\hat{L}$ indicates the original contents of matrix L. The predicate in Step 1b in Fig. 2 indicates the desired state upon completion (the postcondition).

Step 2: Determine P_{inv}. In order to determine possible intermediate contents of the matrix L, one starts by partitioning the input and output operands, in this case L and $\hat{L}$. The partitioning corresponds to an assumption that algorithms progress through data in a systematic fashion. Since L is lower triangular, it becomes important to partition it into four quadrants,

Step	Annotated Algorithm: $L := L^{-1}$
1a	$\left\{ L = \hat{L} \wedge \text{LowerTri}(L) \right\}$
4	**Partition** $L = \left(\begin{array}{cc} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right)$ and $\hat{L} = \left(\begin{array}{cc} \hat{L}_{TL} & 0 \\ \hline\hline \hat{L}_{BL} & \hat{L}_{BR} \end{array}\right)$ **where** L_{TL} and $\hat{L}_{TL}$ are 0×0
2	$\left\{ \left(\begin{array}{cc} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) = \left(\begin{array}{cc} L_{TL}^{-1} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) \right\}$
3	**while** $\neg\text{SameSize}(L, L_{TL})$ **do**
2,3	$\left\{ \left(\left(\begin{array}{cc} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) = \left(\begin{array}{cc} L_{TL}^{-1} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) \right) \wedge (\neg\text{SameSize}(L, L_{TL})) \right\}$
5a	**Determine block size** b **Repartition** $\left(\begin{array}{cc} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{ccc} L_{00} & 0 & 0 \\ \hline\hline L_{10} & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right)$ and $\left(\begin{array}{cc} \hat{L}_{TL} & 0 \\ \hline\hline \hat{L}_{BL} & \hat{L}_{BR} \end{array}\right) \rightarrow \left(\begin{array}{ccc} \hat{L}_{00} & 0 & 0 \\ \hline\hline \hat{L}_{10} & \hat{L}_{11} & 0 \\ \hline \hat{L}_{20} & \hat{L}_{21} & \hat{L}_{22} \end{array}\right)$ **where** L_{11} and $\hat{L}_{11}$ are $b \times b$
6	$\left\{ \left(\begin{array}{ccc} L_{00} & 0 & 0 \\ \hline\hline L_{10} & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right) = \left(\begin{array}{ccc} \hat{L}_{00}^{-1} & 0 & 0 \\ \hline\hline \hat{L}_{10} & \hat{L}_{11} & 0 \\ \hline \hat{L}_{20} & \hat{L}_{21} & \hat{L}_{22} \end{array}\right) \right\}$
8	$L_{10} := -L_{11}^{-1} L_{10} L_{00}$ $L_{11} := L_{11}^{-1}$
7	$\left\{ \left(\begin{array}{ccc} L_{00} & 0 & 0 \\ \hline L_{10} & L_{11} & 0 \\ \hline\hline L_{20} & L_{21} & L_{22} \end{array}\right) = \left(\begin{array}{ccc} \hat{L}_{00}^{-1} & 0 & 0 \\ \hline -\hat{L}_{11}^{-1} \hat{L}_{10} \hat{L}_{00}^{-1} & \hat{L}_{11}^{-1} & 0 \\ \hline\hline \hat{L}_{20} & \hat{L}_{21} & \hat{L}_{22} \end{array}\right) \right\}$
5b	**Continue with** $\left(\begin{array}{cc} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{ccc} L_{00} & 0 & 0 \\ \hline L_{10} & L_{11} & 0 \\ \hline\hline L_{20} & L_{21} & L_{22} \end{array}\right)$ and $\left(\begin{array}{cc} \hat{L}_{TL} & 0 \\ \hline\hline \hat{L}_{BL} & \hat{L}_{BR} \end{array}\right) \leftarrow \left(\begin{array}{ccc} \hat{L}_{00} & 0 & 0 \\ \hline \hat{L}_{10} & \hat{L}_{11} & 0 \\ \hline\hline \hat{L}_{20} & \hat{L}_{21} & \hat{L}_{22} \end{array}\right)$
2	$\left\{ \left(\begin{array}{cc} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) = \left(\begin{array}{cc} L_{TL}^{-1} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) \right\}$
	enddo
2,3	$\left\{ \left(\left(\begin{array}{cc} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) = \left(\begin{array}{cc} L_{TL}^{-1} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) \right) \wedge \neg\,(\neg\text{SameSize}(L, L_{TL})) \right\}$
1b	$\left\{ L = \hat{L}^{-1} \right\}$

Fig. 2. Worksheet for developing an algorithm for symmetric matrix multiplication

$$L \rightarrow \left(\begin{array}{cc} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right),$$

where L_{TL} and L_{BR} are square so that these diagonal blocks are lower triangular. Here the indices T, B, L, and R stand for Top, Bottom, Left, and Right, respectively.

Now, this partitioned matrix is substituted into the postcondition after which algebraic manipulation expresses the desired final contents of the quadrants in terms of operations with the original contents of those quadrants:

$$\left(\begin{array}{c||c} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) = \left(\begin{array}{c||c} \hat{L}_{TL} & 0 \\ \hline\hline \hat{L}_{BL} & \hat{L}_{BR} \end{array}\right)^{-1} = \left(\begin{array}{c||c} \hat{L}_{TL}^{-1} & 0 \\ \hline\hline -\hat{L}_{BR}^{-1}\hat{L}_{BL}\hat{L}_{TL}^{-1} & \hat{L}_{BR}^{-1} \end{array}\right).$$

At an intermediate stage (at the top of the loop-body) only some of the operations will have been performed. For example, the intermediate state

$$\left(\begin{array}{c||c} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) = \left(\begin{array}{c||c} \hat{L}_{TL}^{-1} & 0 \\ \hline\hline \hat{L}_{BL} & \hat{L}_{BR} \end{array}\right)$$

comes from assuming that only L_{TL} has been updated with the final result while the other parts of the matrix have not yet been touched. Let us use this example for the remainder of this discussion: it becomes P_{inv} in the worksheet in Fig. 1 as illustrated in Fig. 2.

Step 3: Determine Loop-Guard G. We are assuming that after the loop completes, $P_{\text{inv}} \wedge \neg G$ holds. Thus, by choosing a loop-guard G such that $(P_{\text{inv}} \wedge \neg G) \Rightarrow P_{\text{post}}$, it is guaranteed that the loop completes in a state that implies that the desired result has been computed. Notice that when L_{TL} equals all of L,

$$\left(\left(\begin{array}{c||c} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) = \left(\begin{array}{c||c} \hat{L}_{TL}^{-1} & 0 \\ \hline\hline \hat{L}_{BL} & \hat{L}_{BR} \end{array}\right) \wedge \text{SameSize}(L, L_{TL})\right) \Rightarrow (L = \hat{L}^{-1}).$$

Here the predicate SameSize(L, L_{TL}) is *true* iff the dimensions of L and L_{TL} are equal. Thus, the iteration should continue as long as $\neg$SameSize(L, L_{TL}), the loop-guard G in the worksheet.

Step 4: Determine the Initialization. The loop-invariant must hold before entering the loop. Ideally, only the partitioning of operands is required to attain this state. Notice that the initial partitionings given in Step 4 of Fig. 2 result in an L that contains the desired contents, without requiring any update to the contents of L.

Step 5: Determine How to Move Boundaries. Realize that as part of the initialization L_{TL} is 0×0, while upon completion of the loop this part of the matrix should correspond to the complete matrix. Thus, the boundaries, denoted by the double lines, must be moved forward as part of the body of the loop, adding rows and columns to L_{TL}. The approach is to identify parts of the matrix that must be moved between regions at the top of the loop body, and adds them to the appropriate regions at the bottom of the loop body, as illustrated in Steps 5a and 5b in Fig. 2.

Step 6: Determine P_{before}. Notice that the loop-invariant is *true* at the top of the loop body, and is thus *true* after the repartitioning that identifies parts of the matrices to be moved between regions. In Step 6 in Fig. 2 the state, in terms of the repartitioned matrices, is given.

```
Partition $L = \left(\begin{array}{c||c} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right)$
    where $L_{TL}$ is $0 \times 0$

while $\neg$SameSize$(L, L_{TL})$ do
    Determine block size $b$
    Repartition
        $\left(\begin{array}{c||c} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} L_{00} & 0 & 0 \\ \hline\hline L_{10} & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right)$
            where $L_{11}$ is $b \times b$
    $L_{10} := -L_{11}^{-1} L_{10} L_{00}$
    $L_{11} := L_{11}^{-1}$
    Continue with
        $\left(\begin{array}{c||c} L_{TL} & 0 \\ \hline\hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} L_{00} & 0 & 0 \\ \hline L_{10} & L_{11} & 0 \\ \hline\hline L_{20} & L_{21} & L_{22} \end{array}\right)$
enddo
```

Fig. 3. Final algorithm

Step 7: Determine P_{after}. Notice that after the regions have been redefined, (as in Step 5b in Fig. 2), the loop-invariant must again be *true*. Given the redefinition of the regions in Step 5b, the loop-invariant, with the appropriate substitution of what the regions will become, must be *true* after the movement of the double lines. Thus,

$$\left(\begin{array}{c||c} \hat{L}_{TL}^{-1} & 0 \\ \hline\hline \hat{L}_{BL} & \hat{L}_{BR} \end{array}\right) = \left(\begin{array}{c||c} \left(\begin{array}{c|c} \hat{L}_{00} & 0 \\ \hline \hat{L}_{10} & \hat{L}_{11} \end{array}\right)^{-1} & 0 \\ \hline\hline \left(\begin{array}{c|c} \hat{L}_{20} & \hat{L}_{21} \end{array}\right) & \hat{L}_{22} \end{array}\right) = \left(\begin{array}{c|c||c} \hat{L}_{00}^{-1} & 0 & 0 \\ \hline -\hat{L}_{11}^{-1}\hat{L}_{10}\hat{L}_{00}^{-1} & \hat{L}_{11}^{-1} & 0 \\ \hline\hline \hat{L}_{20} & \hat{L}_{21} & \hat{L}_{22} \end{array}\right)$$

must be *true* after the movement of the double lines.

Step 8: Determine the Update S_U. By comparing the state in Step 6 with the desired state in Step 7, the required update, given in Step 8, can be easily determined.

Final Algorithm. Finally, by noting that $\hat{L}$ was introduced only to denote the original contents of L and is never referenced in the update, the algorithm for computing $L := L^{-1}$ can be stated as in Fig. 3.

Note that the second operation in update S_U requires itself an inversion of a triangular matrix. When $b = 1$, this becomes an inversion of the scalar L_{11}. Thus, the so-called "blocked" version of the algorithm, where $b > 1$, could be implemented by calling an "unblocked" version, where $b = 1$ and $L_{11} := L_{11}^{-1}$ is implemented by an inversion of a scalar.

```
FLA_Part_2x2( L,    &LTL, &LTR,
                    &LBL, &LBR,     0, 0, FLA_TL );

while ( FLA_Obj_length( LTL ) != FLA_Obj_length( L ) ){
  b = min( FLA_Obj_length( LBR ), nb_alg );
  FLA_Repart_2x2_to_3x3(
        LTL, /**/ LTR,       &L00, /**/ &L01, &L02,
     /* ************ */   /* ******************** */
                             &L10, /**/ &L11, &L12,
        LBL, /**/ LBR,       &L20, /**/ &L21, &L22,
        b, b, FLA_BR );
  /*-------------------------------------------------*/

  FLA_Trsm(FLA_LEFT, FLA_LOWER_TRIANGULAR,
           FLA_NO_TRANSPOSE, FLA_NONUNIT_DIAG,
           MINUS_ONE, L11, L10);

  FLA_Trmm(FLA_RIGHT, FLA_LOWER_TRIANGULAR,
           FLA_NO_TRANSPOSE, FLA_NONUNIT_DIAG,
           ONE, L00, L10);

  FLA_TriInv( L11 );

  /*-------------------------------------------------*/
  FLA_Cont_with_3x3_to_2x2(
        &LTL, /**/ &LTR,       L00, L01, /**/ L02,
                               L10, L11, /**/ L12,
     /* *************** */   /* ***************** */
        &LBL, /**/ &LBR,       L20, L21, /**/ L22,
        FLA_TL );
}
```

Fig. 4. C implementation

Alternative Algorithms. The steps we just described allow one to derive alternative algorithms: by applying Steps 3-8 with different loop-invariants one can obtain different variants for the same operation.

4 Implementation and Performance

In order to translate the proven correct algorithm into code, we have introduced APIs for the Mscript [10], C, and Fortran programming languages. The APIs were designed to mirror the algorithms as obtained from the worksheet. This allows for a rapid and direct translation to code, reducing chances of coding errors. The C code corresponding to the algorithm in Fig. 3 is illustrated in Fig. 4. The **Partition** statement in the algorithm

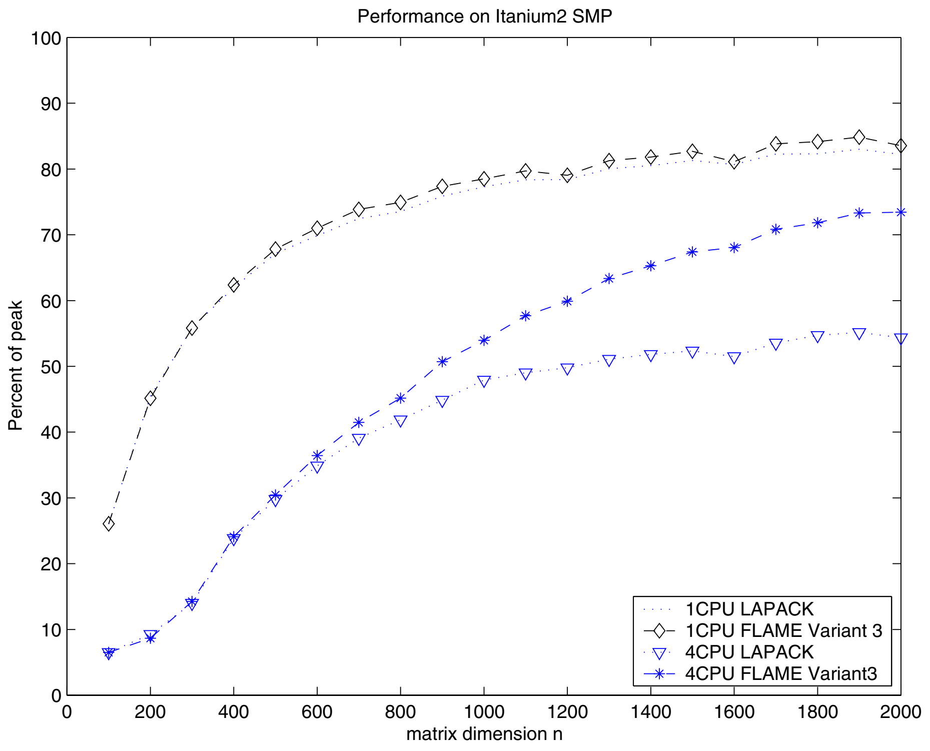


Fig. 5. Performance of LAPACK vs. FLAME on a single CPU and four CPUs of an Itanium2-based SMP. Here Variant3 indicates a different loop invariant which results in an algorithm rich in triangular matrix matrix multiply (TRMM). It is known that the rank-k update is an operation that parallelizes better than TRMM. This explain the better peformance of Variant3 with respect to LAPACK

corresponds to lines 1 and 2 in the code; The **Repartition** and **Continue with** statements are coded by lines 6 to 11 and 25 to 30 respectively. Finally, the updates in the body of the loop correspond to lines 14 to 22 in the code.

Performance attained on an SMP system based on the Intel Itanium2 (R) processor is given in Fig. 5. For all the implementations reported, parallelism is achieved through the use of multithreaded BLAS. While the LAPACK implementation uses the algorithm given in Fig. 3, the FLAME one uses a different algorithmic variant that is rich in rank-k updates, which parallelize better with OpenMP. For this experiment, both libraries were linked to a multithreaded BLAS implemented by Kazushige Goto [6].

5 Conclusion

We presented a methodolody for rapidly deriving and implementing algorithms for linear algebra operations. The techniques in this paper apply to operations for which there are algorithms that consist of a simple initialization followed by a loop. While this may appear to be extremely restrictive, the linear algebra libraries community has made tremendous strides towards modularity. As a consequence, almost any operation can be decomposed into operations (linear algebra building blocks) that, on the one hand, are themselves meaningful linear algebra operations and, on the other hand, whose algorithms have the structure given by the algorithm in Fig. 1.

The derivation of algorithms is dictated by eight steps, while the implementation is a direct translation of the algorithm through a number of API's that we have developed. Using PLAPACK [12], a C library based on MPI, even a parallel implementation for a distributed memory architecture closely mirrors the algorithm as represented in Fig. 3 and is no different from the C code shown in Fig. 4.

One final comment about the eight steps necessary to fill the worksheet in: the process is so systematic that we were able to develop a semi-automated system capable of generating one algorithm starting from a loop invariant. In the paper *Automatic Derivation of Linear Algebra Algorithms with Application to Control Theory*, also presented at this conference, we show how to use the system to solve the Triangular Sylvester Equation.

Additional Information

For additional information on FLAME visit

`http://www.cs.utexas.edu/users/flame/`

Acknowledgments

This research is partially supported by NSF grants ACI-0305163 and CCF-0342369. We thank NEC Solutions (America), Inc. for access to the Intel Itanium2 processor SMP used for the performance experiments.

References

1. E. Anderson, Z. Bai, J. Demmel, J. E. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. E. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 1992.
2. Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Transactions on Mathematical Software*, 31(1), March 2005.
3. Paolo Bientinesi, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. Representing linear algebra algorithms in code: The FLAME application programming interfaces. *ACM Transactions on Mathematical Software*, 31(1), March 2005.
4. Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.
5. Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 14(1):1–17, March 1988.

6. Kazushige Goto and Robert A. van de Geijn. On reducing tlb misses in matrix multiplication. Technical Report CS-TR-02-55, Department of Computer Sciences, The University of Texas at Austin, 2002.
7. John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal linear algebra methods environment. *ACM Trans. Math. Soft.*, 27(4):422–455, December 2001.
8. Isak Jonsson. *Recursive Blocked Algorithms, Data Structures, and High-Performance Software for Solving Linear Systems and Matrix Equations*. PhD thesis, Dept. Computing Science, Umeå University, SE-901 87, Sweden., 2003.
9. C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Soft.*, 5(3):308–323, Sept. 1979.
10. C. Moler, J. Little, and S. Bangert. *Pro-Matlab, User's Guide*. The Mathworks, Inc., 1987.
11. Enrique S. Quintana-Ortí and Robert A. van de Geijn. Formal derivation of algorithms: The triangular Sylvester equation. *ACM Transactions on Mathematical Software*, 29(2):218–243, June 2003.
12. Robert A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. The MIT Press, 1997.

Automatic Derivation of Linear Algebra Algorithms with Application to Control Theory

Paolo Bientinesi[1], Sergey Kolos[2], and Robert A. van de Geijn[1]

[1] Department of Computer Sciences, The University of Texas at Austin
{pauldj,rvdg}@cs.utexas.edu
[2] The Institute for Computational Engineering and Sciences, The University of Texas at Austin
skolos@mail.utexas.edu

Abstract. It is our belief that the ultimate automatic system for deriving linear algebra libraries should be able to generate a set of algorithms starting from the mathematical specification of the target operation only. Indeed, one should be able to visit a website, fill in a form with information about the operation to be performed and about the target architectures, click the SUBMIT button, and receive an optimized library routine for that operation even if the operation has not previously been implemented. In this paper we relate recent advances towards what is probably regarded as an unreachable dream. We discuss the steps necessary to automatically obtain an algorithm starting from the mathematical abstract description of the operation to be solved. We illustrate how these steps have been incorporated into two prototype systems and we show the application of one the two systems to a problem from Control Theory: The Sylvester Equation. The output of this system is a description of an algorithm that can then be directly translated into code via API's that we have developed. The description can also be passed as input to a parallel performance analyzer to obtain optimized parallel routines [5].

1 Introduction

In a series of journal papers we have demostrated how formal derivation techniques can be applied to linear algebra to derive provably correct families of high performance algorithms [6,9,3]. In the paper *Rapid Development of High-Performance Linear Algebra Libraries*, also in this volume [2], we describe the FLAME procedure which returns algorithms for linear algebra operations. It is beneficial for the reader to review that paper to better understand the following discussion.

While the procedure ensures the derived algorithm to be correct, it is the application of the procedure itself that is error prone. It involves tedious algebraic manipulations. As the complexity of the operation we want to implement increases, it becomes more cumbersome to perform the procedure by hand.

We want to stress that the procedure does not introduce unnecessary elements of confusion. Operations once deemed "for experts only" can now be tackled by undergraduates, leaving more ambitious problems for the experts. Let us illustrate this concept with a concrete example: the solution to the triangular Sylvester equation $AX + XB = C$. Algorithms for solving the Sylvester equation have been known for

Step	Annotated Algorithm: $[D, E, F, \ldots] = \text{op}(A, B, C, D, \ldots)$
1a	$\{P_{\text{pre}}\}$
4	**Partition** **where**
2	$\{P_{\text{inv}}\}$
3	**while** G **do**
2,3	$\{(P_{\text{inv}}) \wedge (G)\}$
5a	**Repartition** **where**
6	$\{P_{\text{before}}\}$
8	S_U
7	$\{P_{\text{after}}\}$
5b	**Continue with**
2	$\{P_{\text{inv}}\}$
	enddo
2,3	$\{(P_{\text{inv}}) \wedge \neg (G)\}$
1b	$\{P_{\text{post}}\}$

Fig. 1. Worksheet for developing linear algebra algorithms

about 30 years [1]. Nonetheless new variants are still discovered with some regularity and published in journal papers [4,8]. The methodolody we describe in [3] has been applied to this operation yielding a family of 16 algorithms, including the algorithms already known as well as many undiscovered ones [9]. The only complication is that, as part of the derivation, complex matrix expressions are introduced that require simplification, providing an opportunity for algebra mistakes to be made by a human. One of the variants derived in [9], did not return the correct outcome when implemented and executed. Mistakes in simplifying expressions in order to determine the update were only detected when the updates were rederived with the aid of one of the automated systems described next.

2 Automatic Generation of Algorithms

In Figure 1 we show a generic "worksheet" for deriving linear algebra algorithms. The blanks in the worksheet are filled in by following an eight step procedure, generating algorithms for linear algebra operations. A description of the eight steps is given in the paper *Rapid Development of High-Performance Linear Algebra Libraries*, also in this volume [2]. Here we present the steps again, looking at opportunities for automation.

Step 1: Determine P_{pre} and P_{post}. These two predicates are given as part of the specifications for the operation we want to implement; therefore they are the input to an automated system.

Step 2: Determine P_{inv}. Once the operands have been partitioned and substituted into the postcondition, with the aid of mathematical relations and simplifications, we get the expression for the operation we want to implement as function of the exposed submatrices (partitions) of the input operands. We call this expression the *partitioned matrix expression* (PME).

Loop invariants are obtained by selecting a subset of the operations that appear in the PME.

Step 3: Determine Loop-Guard G. The loop invariant describes the contents of output variables at different points of the program. Upon completion of the loop, the loop invariant is true and the double lines (representing the boundaries of how far the computation has gotten) can be expected to reside at the edges of the partitioned operands. These two pieces of information, the loop invariant and the where boundaries are, can be exploited together to automatically deduce a loop-guard. If a loop-guard cannot be found, the selected loop invariant is labelled as infeasible and no further steps are executed.

Step 4: Determine the Initialization. We require the initialization not to involve any computation. It can be observed that by placing the double lines on the boundaries, the precondition implies the loop invariant. This placement can be automated. If such a placement cannot be found, the selected loop invariant is labelled as infeasible and no further steps are executed.

Step 5: Determine How to Move Boundaries. How to traverse through the data is determined by relating the state of the partitioning (after initialization) to the loop-guard. Also, operands with a particular structure (triangular, symmetric matrices) can only be partitioned and traversed in a way that preserves the structure, thus limiting degrees of freedom. This determination can be automated.

Step 6: Determine P_{before}. This predicate is obtained by: 1) applying the substitution rules dictated by the **Repartion** statement to the loop invariant and 2) expanding and simplifying the expressions. Stage 1) is straightforward. Stage 2) requires symbolic computation tools. Mathematica [11] provides a powerful environment for the required algebraic manipulations, facilitating automation.

Step 7: Determine P_{after}. Computing this predicate is like the computation of the state P_{before} except with different substitution rules. In this case the rules are dictated by the **Continue ... with** statement. Therefore automation is possible.

Step 8: Determine the Update S_U. The updates are determined by a comparison of the states P_{before} and P_{after}. Automation of this step is a process that involves a great deal of pattern matching, symbolic manipulations and requires a library of transformation rules for matrix expressions. While we believe that automation for this step is at least partially achievable, we feel that human intervention is desirable to supervise the process. For this step we envision an interactive system that suggests possible updates and drives user through the process.

3 Automated Systems

Initially, for a limited set of target operations, a fully automated system was prototyped. This system took a description of the PME as input and returned all the possible algorithms corresponding to the feasible loop invariants. This system provided the evidence that at least for simple operations all the steps in Section 1 can be automated. The biggest drawback of the system was that there was no easy way for a user to extend its functionality, to include other classes of input matrices, and to deal with more complex operations (involving the computation of the inverse, transpose, or the solution to linear systems).

In a second effort we developed a more interactive tool that guides the user through the derivation process. The input for this system is a loop invariant for the operation under consideration and the output is a partially filled worksheet (see Fig. 2). We designed this system aiming at modularity and generality. Strong evidence now exists that it can be used to semi-automatically generate all algorithms for all operations to which FLAME has been applied in the past. These include all the BLAS operations, all major factorization algorithms, matrix inversion, reductions to condensed forms, and a large number of operations that arise in control theory.

This semi-automated system plays an important role as part of a possible fully automated system: First, it automatically computes the predicates P_{before} and P_{after}. Second, it can express the status P_{after} as a function of P_{before}, thus pointing out to the user the updates to be computed. Finally, it can determine dependencies among the updates, thus avoiding redundant computations and problems with data being overwritten.

Notice that once the operation we want to implement is expressed in terms of partitioned operands, it is feasible to list possible loop invariants. Not all the loop invariants are feasible: not every loop invariant leads to an algorithm for the target operation. In order to decide whether a loop invariant is feasible, some computations needs to be done. The second prototype system can be used as part of a more complete system to test loop invariants and determine whether they are feasible (thus producing an algorithm) or not (thus discarding it).

4 An Example from Control Theory

We illustrate here how the semi-automated system can be used to derive, with little human intervention, a family of algorithms to solve the triangular Sylvester equation. The solution of such an equation is given by a matrix X that satisfies the equality $AX + XB = C$, where A and B are triangular matrices. We use $X = \Omega(A, B, C)$ to indicate that the matrix X is the solution of the Sylvester equation identified by the matrices A, B and C. Without loss of generality, in the following we assume that both matrices A and B are upper triangular.

As we mentioned in the previous section, the semi-automated system takes a loop invariant as input. Loop invariants are obtained from the PME of the operation we are solving. Partitioning the matrices A, B, C and X,

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right), \left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array}\right), \left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array}\right), \left(\begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array}\right),$$

it is then possible to state the PME for X, solution to the triangular Sylvester equation:

$$\left(\begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array}\right) = \left(\begin{array}{c||c} \begin{array}{l} \Omega(A_{TL}, B_{TL}, \\ \quad C_{TL} - A_{TR}X_{BL}) \end{array} & \begin{array}{l} \Omega(A_{TL}, B_{BR}, \\ \quad C_{TR} - A_{TR}X_{BR} - X_{TL}B_{TR}) \end{array} \\ \hline\hline \Omega(A_{BR}, B_{TL}, C_{BL}) & \begin{array}{l} \Omega(A_{BR}, B_{BR}, \\ \quad C_{BR} - X_{BL}B_{TR}) \end{array} \end{array}\right).$$

From this expression a list of loop invariants are systematically generated by selecting a subset of the operations to be performed. The simplest loop invariant is:

$$\left(\begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array}\right) = \left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline \Omega(A_{BR}, B_{TL}, C_{BL}) & C_{BR} \end{array}\right),$$

which identifies a computation of the solution matrix X in which the bottom left quadrant X_{BL} contains the solution of the equation $A_{BR}X_{BL} + X_{BL}B_{TL} = C_{BL}$; the algorithm proceeds by expanding the quadrant X_{BL} in the top-right direction, until X_{BL} includes the whole matrix X and therefore contains the solution (when quadrant X_{BR} coincides with matrix X, matrices A_{BR}, B_{TL} and C_{BL} coincide with A, B and C).

In the remainder of the paper we concentrate on a slightly more complicated loop invariant:

$$\left(\begin{array}{c||c} X_{TL} & X_{TR} \\ \hline\hline X_{BL} & X_{BR} \end{array}\right) = \left(\begin{array}{c||c} C_{TL} - A_{TR}X_{BL} & C_{TR} \\ \hline\hline \Omega(A_{BR}, B_{TL}, C_{BL}) & C_{BR} \end{array}\right),$$

which corresponds to an algorithm where the quadrant X_{BL} contains again the solution of the equation $A_{BR}X_{BR} + X_{BR}B_{TL} = C_{BL}$, and the quadrant X_{TL} is partially updated. Figure 2 shows the output algorithm generated by the automated system for this loop invariant.

The crucial expressions appear in the boxes labeled “Loop invariant before the updates” (LI-B4)

loop invariant before the updates [LI-B4]

$$\begin{array}{c|c|c} -\left(\hat{A}_{02} \cdot \left(\Omega\left(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20}\right)\right)\right) + \hat{C}_{00} & \hat{C}_{01} & \hat{C}_{02} \\ \hline -\left(\hat{A}_{12} \cdot \left(\Omega\left(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20}\right)\right)\right) + \hat{C}_{10} & \hat{C}_{11} & \hat{C}_{12} \\ \hline \Omega\left(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20}\right) & \hat{C}_{21} & \hat{C}_{22} \end{array}$$

and “Loop invariant after the updates” (LI-Aft).

loop invariant after the update [LI-Aft]

$$\begin{array}{c|c|c} -\left(\hat{A}_{01} \cdot \mathbf{AFT}_{1,0}\right) + \mathbf{B4}_{0,0} & -\left(\hat{A}_{01} \cdot \mathbf{AFT}_{1,1}\right) - \hat{A}_{02} \cdot \mathbf{AFT}_{2,1} + \mathbf{B4}_{0,1} & \mathbf{B4}_{0,2} \\ \hline \Omega\left(\hat{A}_{11}, \hat{B}_{00}, \mathbf{B4}_{1,0}\right) & \Omega\left(\hat{A}_{11}, \hat{B}_{11}, -\left(\mathbf{AFT}_{1,0} \cdot \hat{B}_{01}\right) - \hat{A}_{12} \cdot \mathbf{AFT}_{2,1} + \mathbf{B4}_{1,1}\right) & \mathbf{B4}_{1,2} \\ \hline \mathbf{B4}_{2,0} & \Omega\left(\hat{A}_{22}, \hat{B}_{11}, -\left(\mathbf{AFT}_{2,0} \cdot \hat{B}_{01}\right) + \mathbf{B4}_{2,1}\right) & \mathbf{B4}_{2,2} \end{array}$$

These two predicates dictate the computations to be performed in the algorithm. LI-B4 expresses the current contents of the matrix X (i.e. the loop invariant), while LI-Aft indicates what X needs to contain at the bottom of the loop, i.e. -after- the updates. This is to ensure that the selected loop invariant holds at each iteration of the loop. For the sake of readability Fig. 2 presents abbreviated versions of these predicates, and the box “Updates” is left empty. LI-Aft presents a few visual cues to compress the otherwise long and unreadable expressions. We provide here a full description of both predicates and we explicitly state the updates which are encoded in LI-Aft.

Operation: sylv3(A B C)

```
{Precondition: ...}
```

Partition

$$A \rightarrow \begin{pmatrix} \hat{A}_{TL} & \hat{A}_{TR} \\ 0 & \hat{A}_{BR} \end{pmatrix} \qquad B \rightarrow \begin{pmatrix} \hat{B}_{TL} & \hat{B}_{TR} \\ 0 & \hat{B}_{BR} \end{pmatrix} \qquad C \rightarrow \begin{pmatrix} \hat{C}_{TL} & \hat{C}_{TR} \\ \hat{C}_{BL} & \hat{C}_{BR} \end{pmatrix}$$

where ...

loop invariant:

$$\begin{pmatrix} -\left(\hat{A}_{TR} \cdot \left(\Omega\left(\hat{A}_{BR}, \hat{B}_{TL}, \hat{C}_{BL}\right)\right)\right) + \hat{C}_{TL} & \hat{C}_{TR} \\ \Omega\left(\hat{A}_{BR}, \hat{B}_{TL}, \hat{C}_{BL}\right) & \hat{C}_{BR} \end{pmatrix}$$

While ...

Repartition

$$\begin{pmatrix} \hat{A}_{TL} & \hat{A}_{TR} \\ 0 & \hat{A}_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} \begin{pmatrix} \hat{A}_{00} & \hat{A}_{01} \\ 0 & \hat{A}_{11} \end{pmatrix} & \begin{pmatrix} \hat{A}_{02} \\ \hat{A}_{12} \end{pmatrix} \\ 0 & \hat{A}_{22} \end{pmatrix}, \begin{pmatrix} \hat{B}_{TL} & \hat{B}_{TR} \\ 0 & \hat{B}_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} \hat{B}_{00} & \left\{\hat{B}_{01}, \hat{B}_{02}\right\} \\ 0 & \begin{pmatrix} \hat{B}_{11} & \hat{B}_{12} \\ 0 & \hat{B}_{22} \end{pmatrix} \end{pmatrix}, \begin{pmatrix} \hat{C}_{TL} & \hat{C}_{TR} \\ \hat{C}_{BL} & \hat{C}_{BR} \end{pmatrix} \rightarrow \left(\begin{matrix} \begin{pmatrix} \hat{C}_{00} \\ \hat{C}_{10} \end{pmatrix} \\ \hat{C}_{20} \end{matrix} \right.$$

loop invariant before the updates [LI-B4]

$-\left(\hat{A}_{02} \cdot \left(\Omega\left(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20}\right)\right)\right) + \hat{C}_{00}$	$\hat{C}_{01}$	$\hat{C}_{02}$
$-\left(\hat{A}_{12} \cdot \left(\Omega\left(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20}\right)\right)\right) + \hat{C}_{10}$	$\hat{C}_{11}$	$\hat{C}_{12}$
$\Omega\left(\hat{A}_{22}, \hat{B}_{00}, \hat{C}_{20}\right)$	$\hat{C}_{21}$	$\hat{C}_{22}$

UPDATES...

loop invariant after the update [LI-Aft]

$-\left(\hat{A}_{01} \cdot \mathbf{AFT}_{1,0}\right) + \mathbf{B4}_{0,0}$	$-\left(\hat{A}_{01} \cdot \mathbf{AFT}_{1,1}\right) - \hat{A}_{02} \cdot \mathbf{AFT}_{2,1} + \mathbf{B4}_{0,1}$	$\mathbf{B4}_{0,2}$
$\Omega\left(\hat{A}_{11}, \hat{B}_{00}, \mathbf{B4}_{1,0}\right)$	$\Omega\left(\hat{A}_{11}, \hat{B}_{11}, -\left(\mathbf{AFT}_{1,0} \cdot \hat{B}_{01}\right) - \hat{A}_{12} \cdot \mathbf{AFT}_{2,1} + \mathbf{B4}_{1,1}\right)$	$\mathbf{B4}_{1,2}$
$\mathbf{B4}_{2,0}$	$\Omega\left(\hat{A}_{22}, \hat{B}_{11}, -\left(\mathbf{AFT}_{2,0} \cdot \hat{B}_{01}\right) + \mathbf{B4}_{2,1}\right)$	$\mathbf{B4}_{2,2}$

Continue with

$$\begin{pmatrix} \hat{A}_{TL} & \hat{A}_{TR} \\ 0 & \hat{A}_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} \hat{A}_{00} & \left\{\hat{A}_{01}, \hat{A}_{02}\right\} \\ 0 & \begin{pmatrix} \hat{A}_{11} & \hat{A}_{12} \\ 0 & \hat{A}_{22} \end{pmatrix} \end{pmatrix}, \begin{pmatrix} \hat{B}_{TL} & \hat{B}_{TR} \\ 0 & \hat{B}_{BR} \end{pmatrix} \rightarrow \begin{pmatrix} \begin{pmatrix} \hat{B}_{00} & \hat{B}_{01} \\ 0 & \hat{B}_{11} \end{pmatrix} & \begin{pmatrix} \hat{B}_{02} \\ \hat{B}_{12} \end{pmatrix} \\ 0 & \hat{B}_{22} \end{pmatrix}, \begin{pmatrix} \hat{C}_{TL} & \hat{C}_{TR} \\ \hat{C}_{BL} & \hat{C}_{BR} \end{pmatrix} \rightarrow \left(\begin{matrix} \left\{\hat{C}_{00}, \hat{C}_{0}\right. \\ \left(\begin{matrix} \hat{C}_{10} & \hat{C}_{1} \\ \hat{C}_{20} & \hat{C}_{2} \end{matrix}\right. \end{matrix} \right.$$

end while

Fig. 2. Algorithm returned by the semi-automatic system

The complete expression for the predicate LI-B4 is:

$$\left(\begin{array}{c||c|c} X_{00} & X_{01} & X_{02} \\ \hline X_{10} & X_{11} & X_{12} \\ \hline\hline X_{20} & X_{21} & X_{22} \end{array}\right) = \left(\begin{array}{c||c|c} C_{00} - A_{02}X_{20} & C_{01} & C_{02} \\ \hline C_{10} - A_{12}X_{20} & C_{11} & C_{12} \\ \hline\hline \Omega(A_{22}, B_{00}, C_{20}) & C_{21} & C_{22} \end{array}\right),$$

and we refer to the (i, j) quadrant of the right-hand side as $\mathrm{B4}_{ij}$. So for instance $\mathrm{B4}_{20}$ corresponds to the expression $\Omega(A_{22}, B_{00}, C_{20})$.

The complete expression for the predicate LI-Aft is daunting:

$$\left(\begin{array}{c|c||c} X_{00} & X_{01} & X_{02} \\ \hline\hline X_{10} & X_{11} & X_{12} \\ \hline X_{20} & X_{21} & X_{22} \end{array}\right) =$$

$$\left(\begin{array}{c|c||c} \begin{array}{l} C_{00} - A_{02}\Omega(A_{22}, B_{00}, C_{20}) - \\ A_{01}\Omega(A_{11}, B_{00}, \\ \qquad C_{10} - A_{12}\Omega(A_{22}, B_{00}, C_{20})) \end{array} & \begin{array}{l} C_{01} - \\ A_{01}\Omega\big(A_{11}, B_{11}, C_{11} - \\ \qquad \Omega(A_{11}, B_{00}, C_{10} - A_{12}\Omega(A_{22}, B_{00}, C_{20}))B_{01} - \\ \qquad A_{12}\Omega(A_{22}, B_{11}, C_{21} - \Omega(A_{22}, B_{00}, C_{20})B_{01})\big) - \\ A_{02}\Omega(A_{22}, B_{11}, C_{21} - \Omega(A_{22}, B_{00}, C_{20})B_{01}) \end{array} & C_{02} \\ \hline\hline \begin{array}{l} \Omega(A_{11}, B_{00}, \\ \quad C_{10} - A_{12}\Omega(A_{22}, B_{00}, C_{20})) \end{array} & \begin{array}{l} \Omega\big(A_{11}, B_{11}, C_{11} - \\ \quad \Omega(A_{11}, B_{00}, C_{10} - A_{12}\Omega(A_{22}, B_{00}, C_{20}))B_{01} - \\ \quad A_{12}\Omega(A_{22}, B_{11}, C_{21} - \Omega(A_{22}, B_{00}, C_{20})B_{01})\big) \end{array} & C_{12} \\ \hline \Omega(A_{22}, B_{00}, C_{20}) & \Omega(A_{22}, B_{11}, C_{21} - \Omega(A_{22}, B_{00}, C_{20})B_{01}) & C_{22} \end{array}\right),$$

and the quadrants in the right-hand side of LI-Aft are identified by

$$\begin{array}{c|c|c} \mathrm{Aft}_{00} & \mathrm{Aft}_{01} & \mathrm{Aft}_{02} \\ \hline \mathrm{Aft}_{10} & \mathrm{Aft}_{11} & \mathrm{Aft}_{12} \\ \hline \mathrm{Aft}_{20} & \mathrm{Aft}_{21} & \mathrm{Aft}_{22} \end{array}.$$

Once the predicates LI-B4 and LI-Aft are known, we are one step away to have a complete algorithm. The updates S_U remain to be computed (Step 8 in Section 2). S_U are statements to be executed in a state in which the predicate LI-B4 holds and they ensure that upon termination the predicate LI-Aft holds.

In this example, given the complexity of the expressions, such a task can be challenging even for experts, and is definitely prone to errors. A (semi-)automated system is useful, if not indispensable. Our system has a number of features to make Step 8 (discovering the updates S_U) as simple as possible:

– The expressions in LI-Aft are scanned to detect quantities contained in LI-B4. Such quantities are currently stored and therefore available to be used as operands; they are identified by boxed grey highlighting. For example, recognizing that the expression $\Omega(A_{22}, B_{00}, C_{20})$ is contained in the quadrant $\mathrm{B4}_{20}$, the system would always display it as: $\boxed{\Omega(A_{22}, B_{00}, C_{20})}$

– A quantity currently available (therefore higlighted) can be replaced by a label indicating the quadrant that contains it. This feature helps to shorten complicated expressions. As an example, one instance of $\Omega(A_{22}, B_{00}, C_{20})$ would be replaced by $\boxed{\mathrm{B4}_{20}}$. Notice that $\Omega(A_{22}, B_{00}, C_{20})$ appears in the quadrant Aft_{00}, as part of the expression $C_{00} - A_{02}\Omega(A_{22}, B_{00}, C_{20})$; in this case the instance is not replaced by $\boxed{\text{B4}_{20}}$ because the entire (and more complex) expression is recognized to appear in B4_{00}. Therefore, $C_{00} - A_{02}\Omega(A_{22}, B_{00}, C_{20})$ is displayed as $\boxed{\text{B4}_{00}}$.

- Dependencies among quadrants are investigated. If the same computation appears in two or more quadrants, the system imposes an ordering to avoid redundant computations. Example: the quadrant Aft_{00}, after the replacements explained in the former two items, would look like $\boxed{\text{B4}_{00}} - A_{01}\Omega\big(A_{11}, B_{00}, C_{10} - A_{12}\boxed{\text{B4}_{20}}\big)$, and recognizing that the quantity $\Omega\big(A_{11}, B_{00}, C_{10} - A_{12}\boxed{\text{B4}_{20}}\big)$ is what the quadrant Aft_{10} has to contain at the end of the computation, the system leaves such an expression unchanged in quadrant Aft_{10} and instead replaces it in quadrant Aft_{00}, which would then be displayed as $\boxed{\text{B4}_{00}} - A_{01}\boxed{\text{Aft}_{10}}$.

The repeated application of all these steps yields a readable expression for LI-Aft, as shown in Figure 2. The updates are encoded in LI-Aft and can be made explicit by applying the following simple rules:

- The assignments are given by the componentwise assignment

$$\begin{array}{c|c|c} X_{00} & X_{01} & X_{02} \\ \hline X_{10} & X_{11} & X_{12} \\ \hline X_{20} & X_{21} & X_{22} \end{array} := \begin{array}{c|c|c} \text{Aft}_{00} & \text{Aft}_{01} & \text{Aft}_{02} \\ \hline \text{Aft}_{10} & \text{Aft}_{11} & \text{Aft}_{12} \\ \hline \text{Aft}_{20} & \text{Aft}_{21} & \text{Aft}_{22} \end{array}.$$

  In our example it results:

$$\begin{array}{c|c|c} X_{00} & X_{01} & X_{02} \\ \hline X_{10} & X_{11} & X_{12} \\ \hline X_{20} & X_{21} & X_{22} \end{array} :=$$

$$\begin{array}{c|c|c} \boxed{\text{B4}_{00}} - A_{01}\boxed{\text{Aft}_{10}} & \boxed{\text{B4}_{01}} - A_{01}\boxed{\text{Aft}_{11}} - A_{02}\boxed{\text{Aft}_{21}} & \boxed{\text{B4}_{02}} \\ \hline \Omega(A_{11}, B_{00}, \boxed{\text{B4}_{10}}) & \Omega(A_{11}, B_{11}, \boxed{\text{B4}_{11}} - \boxed{\text{Aft}_{10}}B_{01} - A_{12}\boxed{\text{Aft}_{21}}) & \boxed{\text{B4}_{12}} \\ \hline \boxed{\text{B4}_{20}} & \Omega(A_{22}, B_{11}, \boxed{\text{B4}_{21}} - \boxed{\text{Aft}_{20}}B_{01}) & \boxed{\text{B4}_{22}} \end{array}$$

- Every assignment of the form $X_{ij} = \boxed{\text{B4}_{ij}}$ corresponds to a no-operation.
- Every assignment whose right-hand side presents one or more operands of the form $\boxed{\text{Aft}_{ij}}$ has to be executed after the quadrant (i, j) has been computed. Once the expression in quadrant (i, j) has been computed, $\boxed{\text{Aft}_{ij}}$ has to be rewritten as X_{ij}.
- Assignments with a right-hand side containing only non-highlighted expressions and/or operands of the form $\boxed{\text{B4}_{ij}}$ can be computed immediately.

A valid set of updates for the current example is given by:

$$\begin{aligned} X_{10} &:= \Omega(A_{11}, B_{00}, \boxed{\text{B4}_{10}}) \\ X_{00} &:= \boxed{\text{B4}_{00}} - A_{01}X_{10} \\ X_{21} &:= \Omega(A_{22}, B_{11}, \boxed{\text{B4}_{21}} - X_{20}B_{01}) \\ X_{11} &:= \Omega(A_{11}, B_{11}, \boxed{\text{B4}_{11}} - X_{10}B_{01} - A_{12}X_{21}) \\ X_{01} &:= \boxed{\text{B4}_{01}} - A_{01}X_{11} - A_{02}X_{21}. \end{aligned}$$

The final algorithm is:

```
Partition $A = \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right), B = \left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array}\right), C = \left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array}\right)$
    where $A_{TL}$ is $0 \times 0$, $B_{BR}$ is $0 \times 0$, $C_{BL}$ is $0 \times 0$

while $\neg$SameSize$(C, C_{BL})$ do
    Determine block size $b_m$ and $b_n$
    Repartition
    $\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline 0 & A_{11} & A_{12} \\ \hline\hline 0 & 0 & A_{22} \end{array}\right), \left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} B_{00} & B_{01} & B_{02} \\ \hline\hline 0 & B_{11} & B_{12} \\ \hline 0 & 0 & B_{22} \end{array}\right)$
    $\left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} C_{00} & C_{01} & C_{02} \\ \hline C_{10} & C_{11} & C_{12} \\ \hline\hline C_{20} & C_{21} & C_{22} \end{array}\right)$
        where $A_{11}$ is $b_m \times b_m$, $B_{11}$ is $b_n \times b_n$, $C_{11}$ is $b_m \times b_n$

    $C_{10} := \Omega(A_{11}, B_{00}, C_{10})$
    $C_{00} := C_{00} - A_{01}C_{10}$
    $C_{21} := \Omega(A_{22}, B_{11}, C_{21} - C_{20}B_{01})$
    $C_{11} := \Omega(A_{11}, B_{11}, C_{11} - C_{10}B_{01} - A_{12}C_{21})$
    $C_{01} := C_{01} - A_{01}X_{11} - A_{02}X_{21}$

    Continue with
    $\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline 0 & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline 0 & A_{11} & A_{12} \\ \hline 0 & 0 & A_{22} \end{array}\right), \left(\begin{array}{c||c} B_{TL} & B_{TR} \\ \hline\hline 0 & B_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} B_{00} & B_{01} & B_{02} \\ \hline 0 & B_{11} & B_{12} \\ \hline\hline 0 & 0 & B_{22} \end{array}\right)$
    $\left(\begin{array}{c||c} C_{TL} & C_{TR} \\ \hline\hline C_{BL} & C_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} C_{00} & C_{01} & C_{02} \\ \hline\hline C_{10} & C_{11} & C_{12} \\ \hline C_{20} & C_{21} & C_{22} \end{array}\right)$
enddo
```

which appears in [9] as Algorithm C2 in Table II.

5 Conclusion

In an effort to demonstrate that automatic derivation of linear algebra algorithm is achievable, we developed two (semi-)automated systems. The first system is fully automated but with very limited scope. It showed that at least for simple operations all steps of the FLAME procedure can be automated. The second system is more interactive and we believe as general as the FLAME approach itself.

The described system has allowed us to automatically generate algorithms for most of the equations in a recent Ph.D. dissertation [7]. In that dissertation, a number of important and challenging linear algebra problems arising in control theory are studied. For most of these problems, one algorithm and implementation is offered. By contrast, with the aid of our automated systems we were able to derive whole families of algorithms and their implementations (in Matlab Mscript as well as in C) collectively in a matter of hours. The implementations yielded correct answers for the first and all inputs with which they were tested. Moreover, parallel implementations can be just as easily created with the FLAME-like extension of our Parallel Linear Algebra Package (PLAPACK) [10].

Additional Information: For additional information on FLAME visit
`http://www.cs.utexas.edu/users/flame/`

Acknowledgments

This research was partially supported by NSF grants ACI-0305163 and CCF-0342369.

References

1. R. H. Bartels and G. W. Stewart. Solution of the matrix equation AX + XB = C. *Commun. ACM*, 15(9):820–826, 1972.
2. Paolo Bientinesi, John A. Gunnels, Fred G. Gustavson, Greg M. Henry, Margaret E. Myers, Enrique S. Quintana-Orti, and Robert A. van de Geijn. Rapid development of high-performance linear algebra libraries. In *Proceedings of PARA'04 State-of-the-Art in Scientific Computing*, June 20-23 2004. To appear.
3. Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Ortí, and Robert A. van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Transactions on Mathematical Software*, 31(1), March 2005.
4. Bo Kågström and Peter Poromaa. Lapack-style algorithms and software for solving the generalized Sylvester equation and estimating the separation between regular matrix pairs. *ACM Transactions on Mathematical Software*, 22(1):78–103, 1996.
5. John Gunnels. *A Systematic Approach to the Design and Analysis of Parallel Dense Linear Algebra Algorithms*. PhD thesis, The University of Texas at Austin, Department of Computer Sciences. Technical Report CS-TR-01-44, December 2001.
6. John A. Gunnels, Fred G. Gustavson, Greg M. Henry, and Robert A. van de Geijn. FLAME: Formal linear algebra methods environment. *ACM Transactions on Mathematical Software*, 27(4):422–455, December 2001.
7. Isak Jonsson. *Recursive Blocked Algorithms, Data Structures, and High-Performance Software for Solving Linear Systems and Matrix Equations*. PhD thesis, Dept. Computing Science, Umeå University, SE-901 87, Sweden., 2003.
8. Isak Jonsson and Bo Kågström. Recursive blocked algorithms for solving triangular systems—part i: one-sided and coupled Sylvester-type matrix equations. *ACM Transactions on Mathematical Software*, 28(4):392–415, 2002.
9. Enrique S. Quintana-Ortí and Robert A. van de Geijn. Formal derivation of algorithms: The triangular Sylvester equation. *TOMS*, 29(2):218–243, June 2003.
10. Robert A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. MIT Press '97.
11. Stephen Wolfram. *The Mathematica Book: 3rd Edition*. Cambridge University Press, 1996.

Cluster Computing for Financial Engineering

Shirish Chinchalkar, Thomas F. Coleman, and Peter Mansfield

Cornell Theory Center
Cornell University
55 Broad Street, Third Floor
New York, NY 10004, USA
{shirish,coleman,peterm}@tc.cornell.edu

Abstract. The pricing of a portfolio of financial instruments is a common and important computational problem in financial engineering. In addition to pricing, a portfolio or risk manager may be interested in determining an effective hedging strategy, computing the value at risk, or valuing the portfolio under several different scenarios. Because of the size of many practical portfolios and the complexity of modern financial instruments the computing time to solve these problems can be several hours. We demonstrate a powerful and practical method for solving these problems on clusters using web services.

1 Introduction

The problems of financial engineering, and more generally computational finance, represent an important class of computationally intensive problems arising in industry. Many of the problems are portfolio problems. Examples include: determine the fair value of a portfolio (of financial instruments), compute an effective hedging strategy, calculate the value-at-risk, and determine an optimal rebalance of the portfolio. Because of the size of many practical portfolios, and the complexity of modern financial instruments, the computing time to solve these problems can be several hours.

Financial engineering becomes even more challenging as future 'scenarios' are considered. For example, hedge fund managers must peer into the future. How will the value of my portfolio of convertibles change going forward if interest rates climb but the underlying declines, and volatility increases? If the risk of default of a corporate bond issuer rises sharply over the next few years, how will my portfolio valuation be impacted? Can I visualize some of these dependencies and relationships evolving over the next few years? Within a range of parameter fluctuations, what is the worst case scenario?

Clearly such "what if" questions can help a fund manager decide today on portfolio adjustments and hedging possibilities. However, peering into the future can be very expensive. Even "modest" futuristic questions can result in many hours of computing time on powerful workstations. The obvious alternative to waiting hours (possibly only to discover that a parameter has been mis-specified), is to move the entire portfolio system to a costly supercomputer. This is a cumbersome, inefficient, and "user unfriendly" approach. However, there is good news: most of these practical problems represent loosely-coupled computations and can be solved efficiently on a cluster of processors in a master-worker framework.

We have been investigating the design of effective parallel approaches to the problems of financial engineering, and computational finance, on clusters of servers using web services. Our particular approach is to represent the portfolio in Excel with the back-end computing needs satisfied by a cluster of industry standard processors running in web services mode. The user environment we have used is Microsoft's .NET.

2 Introduction to Web Services

A web service is a piece of functionality, such as a method or a function call, exposed through a web interface ([1]). Any client on the internet can use this functionality by sending a text message encoded in XML to a server, which hosts this functionality. The server sends the response back to the client through another XML message. For example, a web service could compute the price of an option given the strike, the stock price, volatility, and interest rate. Any application over the internet could invoke this web service whenever it needs the price of such an option. There are several advantages in using web services to perform computations:

1. XML and HTTP are industry standards. So, we can write a web service in Java on Linux and invoke it from a Windows application written in C# and vice a versa.
2. Using Microsoft's .NET technology, we can invoke web services from office applications such as Microsoft Excel. This feature is especially useful in the financial industry, since a lot of end-user data is stored in Excel spreadsheets.
3. No special-purpose hardware is required for running web services. Even different types of computers in different locations can be used together as a web services cluster.
4. Since the web service resides only on the web server(s), the client software does not need to be updated every time the web service is modified. (However, if the interface changes, the client will need to be updated).
5. The web service code never leaves the server, so proprietary code can be protected.
6. Web services can be accessed from anywhere. No special purpose interface is necessary. Even a hand-held device over a wireless network and the internet can access web services.

In the context of large-scale financial computations, there are some limitations to the utilization of web services as well:

1. There is no built-in mechanism to communicate with other web services. This limits the use to loosely coupled applications.
2. The results of any computation performed during a single web service call can only be sent to the client at the end of that web service call. Thus, there is no mechanism for sending partial results while the computation is going on, without using another technology such as MPI.
3. Since messages are sent using a text format over the internet, this is not a viable computational technique for "short" computations involving a lot of data to be communicated.

2.1 A Simple Web Service

The following code shows a web service which adds two integers. It is written using the C# programming language. This code is written in the form of a class. When compiled it produces a dll which can be installed on the web server.

```
using System;
using System.Web.Services;
using System.Web.Services.Protocols;

namespace testnamespace
{
    public class testcls : System.Web.Services.WebService
    {
        [WebMethod]
        public int add(int a, int b)
        {
            return a + b;
        }
    }
}
```

This example shows that writing a web service is no different from writing a function or a method that performs the same computation. Other than a couple of declarative statements, there is no difference between a web service and an ordinary function. Notice that there is no reference to message passing, converting data into XML, and so on. These details are hidden from the programmer.

2.2 A Simple Main Program

The following code shows a main program which accesses this web service and adds two numbers.

```
using System;
using testclient.localhost;

namespace testclient
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
            testcls t = new testcls();
            int a, b, c;
            a = 1;
            b = 2;
            c = t.add(a, b);
            Console.WriteLine("{0} + {1} = {2}", a, b, c);
        }
    }
}
```

Again, from this main program, it is evident that invoking the web service is no different from making a function call within the same process. Only one additional statement refers to the web service at the top of the program.

2.3 Cluster Computing Using Web Services

A typical portfolio manager could have a large portfolio of complex instruments. These instruments may have to be priced every day. Often, several future scenarios of the stock market or interest rates may have to be simulated and the instruments may have to be priced in each scenario. Clearly, a lot of computing power is necessary. If the instruments can be priced independently of one another, we can make use of web services to perform this computation.

The entire computation can be partitioned into several tasks. Each task can consist of the pricing of a single instrument. We can have a separate web service to price each instrument. The client then simply needs to invoke the appropriate web service for each instrument. We can use other models of computation as well. For instance, in case of Monte Carlo simulation, we could split the simulations among the processors.

Figure 1 shows the overall organization of our architecture. The front-end is a typical laptop or a desktop running Excel. Data related to the portfolio is available in an Excel spreadsheet. This front-end is connected over internet or a LAN to a cluster of nodes, each of which runs a web server. When a large computation is to be performed, it is broken into smaller tasks by the Excel front-end. Each task is then shipped to an individual node which works on it independent of the other nodes. The nodes send results back to Excel, which is used to view results.

3 Load Balancing

Given a set of tasks, we can distribute them across a .NET web services cluster in two different ways. We could send all the tasks at once to the main cluster node which uses Network Load Balancing (NLB) to distribute the tasks. However, the NLB monitors network traffic and considers those nodes that are actively communicating as busy and those that are not as idle. This is reasonable in transaction processing applications where each task can be processed quickly and the number of tasks is very large. For the problems we are interested in, we have a relatively small number of tasks, each of which takes seconds, minutes, or hours of computational time. For such problems, a node which is not sending messages might be busy doing computation and might be wrongly classified as idle by NLB. In such cases, the following approach is more suitable: We first assign tasks, one to each processor. When any task finishes, the next task is sent to the node

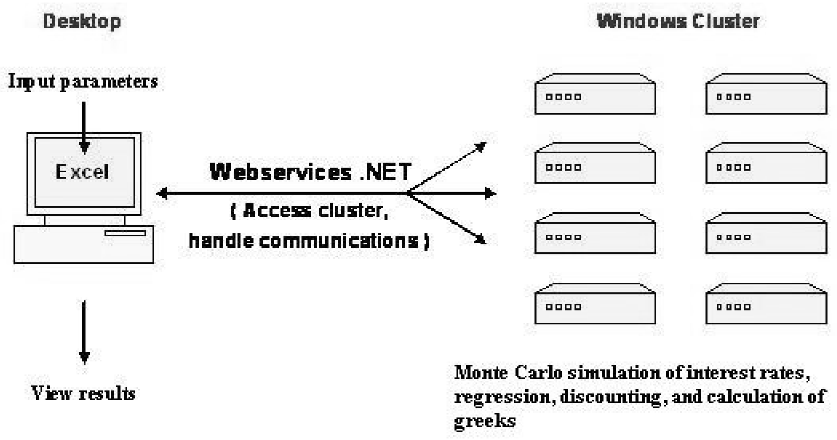

Desktop
Windows Cluster
Input parameters
Excel
Webservices .NET
(Access cluster, handle communications)
View results
Monte Carlo simulation of interest rates, regression, discounting, and calculation of greeks

Fig. 1. Overview of computing architecture

which finished that task. This algorithm works well in practice provided there is only one application running on the cluster. If multiple applications need to run simultaneously, a centralized manager is necessary.

The load balancing mechanism described above can be implemented as a class shown below, provided all data for all tasks is known before any task is executed. Fortunately, most finance applications that involve pricing portfolios of instruments fall in this category. By using a load balancing template, we can remove from the user application, most of the low-level "plumbing" related to multi-threaded operation. This makes applications significantly easier to program and promotes code reuse.

The following pseudo-code shows how such a load balancing class can be used inside a C# client application:

```
allargs = Array.CreateInstance(typeof(appinclass),
                                          numprobs);
for (int i=0; i<numprobs; i++)
{
    appinclass argtemp = new appinclass();

    // set arguments here
    // . . .
    // . . .

    allargs.SetValue(argtemp, i);
}
LBclass lb = new LBclass();
lb.allargs = allargs;
lb.serverURL = "http://hostname/webservice.asmx";
lb.numnodes = 4;
lb.run();

// wait for results
while (!lb.done) Thread.Sleep(50);
```

All code related to invoking the web service asynchronously on a multi-node cluster, determining free nodes, using locks for multi-threaded operation, sending inputs, receiving results, and generating timing and speedup information is handled by the class LBclass. If the user wishes to process results as they are returned, he needs to write an application-specific callback, which is not shown above. Again, this callback does not involve any lower-level message passing related code.

4 An Example

Cluster computing using web services as outlined above can be used to price portfolios comprising different types of instruments such as risky bonds, convertible bonds, and exotic options. We give an example which involves pricing a portfolio of callable bonds.

A typical corporate bond has a face value, a fixed coupon, and a maturity date. Such a bond pays a fixed amount of interest semi-annually until maturity. At maturity, the face value or principal is returned[3]. A callable bond has an additional feature - the bond may be ‘called back’ by the issuing company by offering the bond holder or the investor an amount equal to the face value of the bond. This buy-back can be made on any of the coupon payment dates. Whether it is optimal for the issuing company to call in the bond or not depends on the prevailing interest rates and predictions of future interest rates. For example, if interest rates drop, it may be in the best interests of the issuing company to buy back the bond. If interest rates are high, the issuing company is unlikely to call in the bond. This presents two problems - first, future interest rates must be simulated, and second, the decision to buy the bond or not should be made at each coupon date, depending on the prevailing interest rate and the prediction of future interest rates.

For this work, we have used the Vasicek model for simulating interest rates. In this model, changes in interest rates are given by the formula

$$dr = a(\bar{r} - r)dt + \sigma dW \tag{4.1}$$

where dr is the change in the interest rate in a small time interval, dt, a is the mean reversion rate, $\bar{r}$ is the mean reversion level, and σ is the volatility. dW is a small increment of the Brownian motion, W (see [4] for more details). Given an initial interest rate, r_0, we can easily simulate future interest rates using the above equation. For valuation of callable bonds and the calculation of greeks (see below), we need several tens of thousands of simulations.

Optimal exercise along each interest rate path is determined using the Least Squares Monte Carlo algorithm, which involves the solution of a linear regression problem at each coupon date and discounting of cashflows along each interest rate path. Details of this algorithm can be found in Longstaff and Schwartz[2].

We illustrate a few additional computations for a single bond. They can be extended to a portfolio quite easily. Along with the price of the bond, we also want the bond's 'greeks'; for example bond delta and bond gamma. Delta is the first derivative of the bond price with respect to the initial interest rate ($\partial B/\partial r$) and gamma is the second derivative of the bond price with respect to the initial interest rate ($\partial^2 B/\partial r^2$), where B is the price of the bond. In this work, we have computed them using finite differences as follows

$$\varDelta = \left.\frac{\partial B}{\partial r}\right|_{r=r_0} \approx \frac{B(r_0+dr)-B(r_0-dr)}{2dr} \tag{4.2}$$

$$\varGamma = \left.\frac{\partial^2 B}{\partial r^2}\right|_{r=r_0} \approx \frac{B(r_0+dr)-2B(r_0)+B(r_0-dr)}{dr^2} \tag{4.3}$$

The above calculations require the pricing of the bond at two additional interest rates, r_0+dr and r_0-dr. For all three pricing runs, we use the same set of random numbers to generate the interest rate paths (see [4]).

Once the greeks are computed, we can approximate the variation of the bond price by the following quadratic

$$B(r) \approx B(r_0) + \varDelta(r-r_0) + \frac{1}{2}\varGamma(r-r_0)^2 \tag{4.4}$$

A risk manager would be interested in knowing how much loss this bond is likely to make, say, 1 month from now. This can be characterized by two metrics: Value at Risk (VaR) and Conditional Value at Risk (CVaR). These can be computed from the above approximation by another Monte Carlo simulation. For an introduction to VaR and CVar see [4].

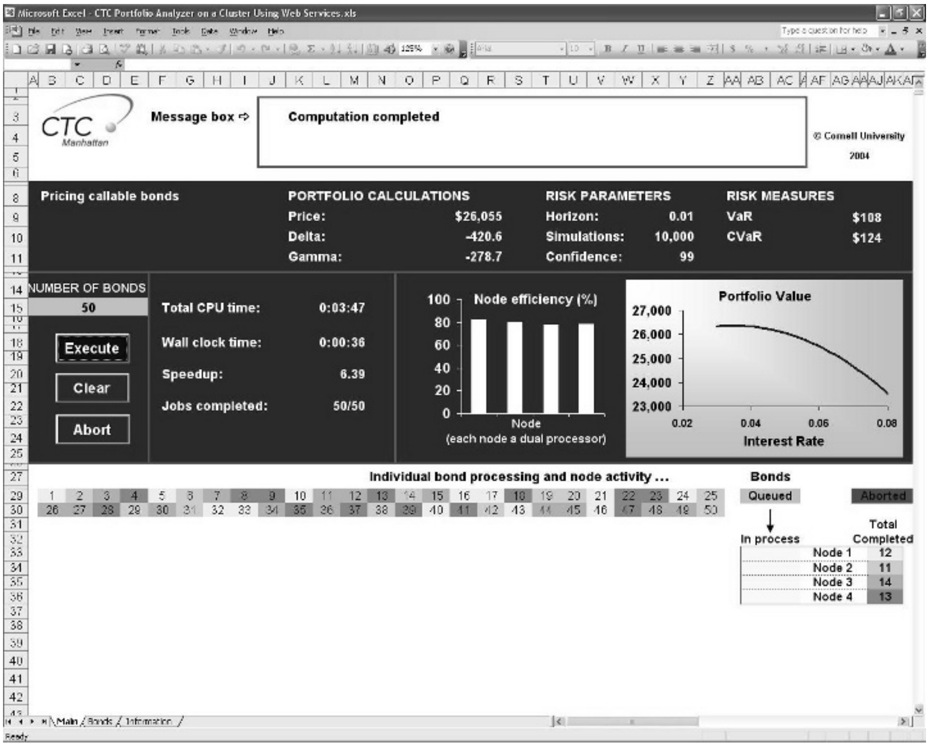
The portfolio price, V, is simply a linear function of the individual bond prices

$$V = \sum_{1}^{n} w_i B_i \tag{4.5}$$

where the portfolio consists of n bonds, with w_i number of bonds of type i. The greeks can be computed analogously, and VaR and CVaR can be determined easily once the greeks are known.

Figure 2 shows the Excel front-end developed for this example. This interface can be used to view bond computing activity, cluster utilization and efficiency, a plot of portfolio price versus interest rate, and portfolio price, Value at Risk (VaR), Conditional Value at Risk (CVaR), and portfolio delta and gamma.

In our example, the web service computes the bond price and bond greeks, whereas the Excel front-end computes the portfolio price, greeks, VaR, and CVaR. Our experiments with portfolios of as few 50 instruments show that on 8 processors, we get speedups of 6.5 or more. On a 64 processor cluster, we have obtained speedups in excess of 60 for portfolios consisting of 2000 instruments, reducing 9 hours of computation to about 9 minutes.



Message box ⇨ Computation completed

© Cornell University 2004

Pricing callable bonds

PORTFOLIO CALCULATIONS		RISK PARAMETERS		RISK MEASURES	
Price:	$26,055	Horizon:	0.01	VaR	$108
Delta:	-420.6	Simulations:	10,000	CVaR	$124
Gamma:	-278.7	Confidence:	99		

NUMBER OF BONDS 50

Total CPU time:	0:03:47
Wall clock time:	0:00:36
Speedup:	6.39
Jobs completed:	50/50

Individual bond processing and node activity ...

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

Bonds: Queued → In process; Aborted

In process	Total Completed
Node 1	12
Node 2	11
Node 3	14
Node 4	13

Fig. 2. Callable bond pricing in Excel

5 Conclusion

Parallel computing, used to speed up a compute-intensive computation, has been under development, and in use by researchers and specialists, for over a dozen years. Because a parallel computing environment is typically an isolated and impoverished one (not to mention very costly!), general industry has been slow to adopt parallel computing technology. Recent web services developments suggest that this situation is now improving, especially for certain application classes, such as portfolio modeling and quantitative analysis in finance. The work we have described here illustrates that a powerful analytic tool can be designed using web services technology to meet some of the computational challenges in computational finance and financial engineering.

Acknowledgements

This research was conducted using resources of the Cornell Theory Center, which is supported by Cornell University, New York State, and members of the Corporate Partnership Program. Thanks to Dave Lifka and his systems group at the Cornell Theory Center (Ithaca) for their support and to the other members of the CTC Computational Finance Group (Yuying Li and Cristina Patron) for their help.

References

1. A. Banerjee, et. al. *C# web services - building web services with .NET remoting and ASP.NET.* Wrox Press Ltd., Birmingham, UK, 2001.
2. F. Longstaff and E. Schwartz. Valuing American Options by Simulation: A Simple Least Squares Approach. *The Review of Financial Studies*, 14:113-147, 2001.
3. Z. Bodie, A. Kane, and A.J. Markus, *Investments*. McGraw Hill. 2001.
4. P. Wilmott. *Paul Wilmott on Quantitative Finance, Volume 2*. John Wiley and Sons, New York, 2000.

Semi-automatic Generation of Grid Computing Interfaces for Numerical Software Libraries*

Erik Elmroth and Rikard Skelander

Dept. of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden
{elmroth,rikard}@cs.umu.se

Abstract. There is an immediate need to develop Grid interfaces for a large set of numerical software libraries, in order to make popular software of today available in the computing infrastructure of tomorrow. As this development work tend to be both tedious and error-prone, this contribution presents a semi-automatic process for generating the interfaces. The underlying principle is to use a front-end tuned for each numerical library and a back-end for each Grid environment considered. Then all library—Grid environment combinations can be generated with a small amount of manual work. The presentation of the main ideas is followed by a proof-of-concept implementation that generates NetSolve interfaces for the complete SLICOT software library, a numerical library comprising nearly 400 Fortran subroutines for numerical computations in the design and analysis of control systems.

Keywords: Grid computing, numerical software libraries, remote computing, interface, SLICOT, NetSolve.

1 Introduction

The rapid development of the Grid computing infrastructure puts strong demands on development of Grid-enabled application software and software libraries. Some of the typical Grid resource usage scenarios are based on the underlying idea that all small computations are performed on a single local computer, while large-scale computations are automatically distributed to appropriate and more powerful computing resources on the Grid. Examples include Grid-empowered problem solving environments and application-oriented web-based Grid portals, so-called science portals.

For both these scenarios, there is an immediate need for interfacing standard software libraries on remote resources. The development of such interfaces tend to be both tedious and error-prone. This contribution presents a semi-automatic process for generating the interfaces, and a proof-of-concept implementation that generates NetSolve interfaces for the complete SLICOT software library.

The front-end of the prototype performs a parsing of calling sequences, variable declarations, and source code documentation in order to automatically determine the subroutine interfaces. Notably, the front-end needs to be tuned to the conventions of

* Financial support has been provided by the Swedish Foundation for Strategic Research under the frame program grant A3 02:128 and The Swedish Research Council (VR) under contract 343-2003-953.

each specific library and the programming language used. Depending on the level of consistency of the documentation, some parts of the calling sequence may not be uniquely determined by the automatic procedure. Such cases are reported and taken care of by hand.

The back-end, which in our proof-of-concept implementation generates so called NetSolve Problem Description Files, is general with respect to which library that has been processed by the front-end, and the interfaces generated are completely portable. As the original SLICOT software includes routines for testing and timing, they can directly be used to verify the correctness of the NetSolve interfaces, by simply calling routines in the NetSolve-enabled library instead of the standard library.

The prototype is demonstrated by generating NetSolve interfaces for the complete SLICOT library. SLICOT is a subroutine library comprising nearly 400 Fortran routines for numerical computations in the design and analysis of control systems.

The outline of the rest of the paper is as follows. The remote computing scenario and the motivation for this work is presented in Section 2. Here, we also exemplify the types of Grid middleware and software libraries considered. The general process for generating interfaces is described in Section 3. The usage of this process in practice, and some lessons learned follows in the description of our proof-of-concept implementation in Section 4. Section 5 briefly describes how to use the NetSolve version of the SLICOT library on more powerful remote resources without having to install more than a small set of NetSolve client software on the local computer. We finally make some concluding remarks in Section 6.

2 Background and Motivation

Several common Grid middleware solutions are based on the underlying principle that data should be sent to some remote resource where the software resides. This principle is often referred to as *remote computing*. In alternative approaches, both data and software are sent from the client machine to a resource (*code shipping*), or both software and data are sent from different locations to a third, computational resource (*proxy computing*).

Our focus is on the generation of interfaces for software libraries in the remote computing scenario. The middleware solution used for our proof-of-concept implementation is NetSolve [3]. NetSolve basically makes it possible to call subroutines on remote systems with interfaces for Fortran, C, Matlab, Mathematica, and Octave (see also Section 4). Similar functionality using alternative interfaces is provided by the Ninf (Network-based information library) software [15], and a recent enhanced version, named Ninf-G [20], that is built on top of standard Globus-based Grid services. Other related efforts include the Purdue Network Computing Hub (PUNCH) [11], Network Enabled Optimization Server (NEOS) [14], the Remote Computing System (RCS) [2], etc. Moreover, the Open Grid Services Architecture [8] defines a more general framework for specifying Grid services based on Web services, that also can be used in the specific scenario outlined here [16].

Application-oriented Grid portals represent another type of Grid environments that in one aspect may have similar requirements. Also a portal may be seen as a remote interface to the software. Given a semi-automated process for generating other Grid interfaces for remote computing, also the set of application-oriented portals may be considered as target. We have already seen both automated efforts and different types of toolkits for constructing application-specific portals, including an automatic tool for Ninf-coupled portals [19] and the SLICOT web portal [7,10].

Our aim is to Grid-enable scientific software libraries typified by LAPACK, SLICOT, Linpack, IBM ESSL [1,4,9,18], and several others. This type of large numerical libraries have been developed for many years, and have proved to be robust, popular and long-lived. Over the years, we have seen that many such libraries, e.g., written in Fortran 77, have continued to stay popular even by users preferring more modern programming languages. Hence, we expect that there will be a demand for using todays numerical libraries also in the future and from the next generation of computing environments, including Grids and web-based Grid portals.

Over time, the relative overhead for transferring the data to a remote resource before performing the computations is decreasing as network performance is increasing more rapidly than the performance of the computers. Today, we see a doubling of network bandwidth every 9–12 months which should be related to the doubling of computer performance every 18 months. As this trend continues, the overhead of data transfer may well be compensated by the reduced manual work for software library installations and tuning.

Clearly, we can foresee the need for at least a number of different interfaces for each of quite a few different libraries, leading to a significant number of library-interface combination. If manually generated one-by-one, without making any attempt to reuse results between the different interfaces generated, we can easily foresee an enormous amount of both tedious and error-prone work.

Hence, we propose to use a semi-automated process where we only need one front-end process for each library and one back-end process for each Grid environment considered, in order to generate all software library—Grid environment combinations requested.

3 Semi-automatic Interface Generation

The interface generation process can basically be viewed as two nearly independent processes performed in sequence. First, a semi-automatic process parses the numerical software library in order to extract all library-specific information required to define the correct calling sequences for all subroutines. This information is then stored in a basic internal format. The second process is the generation of Grid middleware specific interfaces for all subroutines.

The parsing process of the front-end described below is aiming at numerical libraries written in Fortran, as these still are dominating among existing numerical libraries. The general idea, however, is applicable to libraries written in other programming languages as well.

3.1 Front-End: Extracting Software Library Information

The front-end parses the source code of the numerical software library and extracts the required information from the

– Calling sequences.
– Parameter declarations.
– Inline documentation.

The information that can be extracted from the calling sequence is obviously the name and the order of all parameters. The parameter declarations give additional information about the data types, the number of dimensions of array elements, and the leading dimension of at least all but the last dimension of the array arguments. As parameters are passed by reference, the declarations do not have to (and do not in general) contain information about the last dimension of array arguments, which is required to determine the total size of such objects. This information is, however, typically included in the inline documentation of the code. The inline documentation together with the declarations may also tell if the parameters are of type input, output, or both.

The parsing of the inline documentation is by nature more difficult than that of the calling sequence and the parameter declarations, as it does not follow any well-defined syntactical and semantical rules and definitions. Despite this fact, it is typically possible to automatically extract a vast majority of the information required from this documentation by taking advantage of the more or less strict conventions that often are used in state-of-the-art libraries.

If the size of a dimension is expressed in the inline documentation as a constant or as a single input parameter, the automated process has no problems. However, it becomes more complicated if the dimension size is expressed as some function of input parameters. For example, the automatic process may determine from the inline documentation, that a parameter declared as `A(LDA, *)` is to be used as `A(LDA, MAX(M, N))`, where `M` and `N` are included in the list of parameters. In order to perform the communication correctly, both the client and the server resource need to know the exact size the array `A`. Since how to handle this type of issues depends on the functionality of the Grid software, and we strive to make the internal format general with no dependencies on the back-end, we add one extra variable to the internal storage format for each function value requested (e.g., one extra parameter for the function value `MAX(M, N)` in the example above).

There will normally also be cases where the automated process fails to determine, e.g., the size of an array, possibly because its size is expressed in terms of natural language. Such cases are simply identified by a "flag", telling where the user needs to put in some manual work.

The fact that the conventions are different for different libraries makes it necessary to adapt the front-end for each case. Then, the extent to which all routines in a library follows these conventions determines how much manual work is needed after the automatic process.

As the result of the parsing, all information required for each subroutine call is extracted and stored in an internal format. The information includes the subroutine name, the lists of input and output parameters, data types for the parameters, array dimensionality, etc.

3.2 Back-End: Generating Grid Interfaces

The input data for the back-end process is the data generated by the front-end, stored in the internal format. From this data, the back-end can be configured and tuned to automatically generate all the interfaces in the exact format requested for a specific Grid middleware. Of course, this is a process that can be rather different for different Grid middleware, but in most cases the code to be generated has rather limited syntax and semantics which makes the back-end easily developed. Hence, most of this generation is trivial, but we will in the following proof-of-concept implementation also illustrate some technical problems that need to be handled.

Notably, given a back-end for one middleware, it can be used for a automatic translation for any software library for which the requested information is available in the internal format.

4 Proof-of-Concept Demonstration

The feasibility of the process described above have been investigated in a proof-of-concept implementation, that makes the complete SLICOT library available from remote resources via NetSolve.

4.1 Aggregation of Interface Data for the SLICOT Library

SLICOT is a numerical software library for computations in systems and control theory [18], freely available for non-commercial use. The library provides Fortran 77 implementations of algorithms and methods for the design and analysis of control systems. Among the more pronounced design principles for SLICOT is the strive to provide robust, stable and accurate algorithms, to take both memory requirements and floating point performance issues into account, and to follow strict programming and documentation conventions throughout the library.

In total, SLICOT comprises nearly 400 user-callable and computational routines. Around 200 of the routines have associated example programs, example data files and results for illustrations and comparisons. The Basic Linear Algebra Subprograms (BLAS) [5,6,13] and LAPACK [1] are used for underlying linear algebra computations.

SLICOT is organized in eleven groups of routines, depending of their applicability or type:

A:	Analysis Routines
B:	Benchmark and Test Problems
C:	Adaptive Control
D:	Data Analysis
F:	Filtering
I:	Identification
M:	Mathematical Routines
N:	Nonlinear Systems
S:	Synthesis Routines
T:	Transformation Routines
U:	Utility Routines

The parsing of the library has been performed as described in Section 3.1. It should be remarked that the documentation style in SLICOT is not completely homogeneous and does not follow one unified convention as strictly as the SLICOT design goal suggests. This implies some extra efforts in designing and tuning the front-end parser, but it should be noted that almost all information required has been gathered through the automatic process.

The only type of information for which we do not think it is worth the effort to build an automatic tool, is for identifying array dimensions that are not only parameter values. A typical example is to determine the size of an array declared as `WORK(*)`, where the documentation tells that the array is of size `LWORK` and `LWORK` is not a parameter itself but a more complex function of the input parameters. In these cases the automated process simply indicates that manual work is required to add this information.

After completing the automatic parsing and some work by hand, all the requested information is stored in the internal format. Notably, this information is now gathered once and for all and can be use to generate any number of different Grid interfaces using differently configured back-ends. In this proof-of-concept implementation we generate interfaces for NetSolve.

4.2 Generating the NetSolve-SLICOT Interfaces

NetSolve is a Network-enabled solver that gives clients transparent access to software on remote servers [3]. NetSolve *agents* match service requests with the resources available. In order to make software available via NetSolve, each subroutine interface must be specified in a NetSolve Problem Description File (PDF). Based on such files, NetSolve provides interfaces to be called from Fortran, C, Matlab, Mathematica, and Octave. The actual software made available via NetSolve can be written in C or Fortran.

Almost all of the translation from the internal format to the rather basic language of the PDF files can be done automatically. Again, it is only some of the more complicated array dimensions that cause problems. Sizes that are parameter values, e.g., `LDA` in Section 3.1, or unconditional functions of the input parameters, e.g., `LDA*M`, do not cause any problems in the back-end process. For conditional expressions, it is possible to use so called "`@COMP`" expressions in the PDF file. The "`@COMP`" expressions can be used to handle cases, where the array size depends on, e.g., if another parameter is `TRANS = 'Y'` or `TRANS = 'N'` (i.e., if the matrix transpose should be used or not). However, the "`@COMP`" functionality can currently only handle conditional expressions including tests of equality. Apparently, this limitation of the "`@COMP`" expressions will be removed in future NetSolve versions, but for the current version we overcome the problem by adding an extra parameter where the user specifies the array size in these rare cases.

The output from this back-end is one NetSolve PDF-file for each SLICOT subroutine. These files are then to be stored on a server resource that has the NetSolve server software running, and access to the complete SLICOT library including the underlying BLAS and LAPACK routines. In addition to this, the server must register with a NetSolve agent, which then can direct the user's requests to the server.

In order to test the generated interfaces, we have modified the nearly 200 test routines available with SLICOT to call the NetSolve versions or the routines instead of routines in a local SLICOT library. We remark that for libraries where test routines are available, this is in general a very convenient way of testing also the new interfaces.

5 Using SLICOT via NetSolve

In order to use the NetSolve version of SLICOT the user must install the NetSolve client routines on the local computer, set the appropriate environment variable and link with the appropriate NetSolve library. Notably, there is no need to install SLICOT, BLAS, and LAPACK on the local computer as these libraries will only be accessed on remote resources. NetSolve also provides commands for querying an agent about available servers.

With the basic installations in place, there are only marginal changes that have to be made to an application software in order to call the NetSolve version of SLICOT on a remote computer system instead of calling a locally installed SLICOT library. This is illustrated by following small example, showing how a standard SLICOT routine is called via NetSolve from a Fortran or C to application program.

The subroutine head of the original Fortran code for the SLICOT routine SB02MD is outlined below. SB02MD is a routine implementing a Schur vectors method for solving algebraic Riccati equations [12,17,21].

```
SUBROUTINE SB02MD( DICO, HINV, UPLO, SCAL, SORT, N, A,
                   LDA, G, LDG, Q, LDQ, RCOND, WR, WI,
                   S, LDS, U, LDU, IWORK, DWORK,
                   LDWORK, BWORK, INFO )
```

A remote NetSolve call to this routine from a Fortran program is made through a subroutine call to FNETSL with the routine name (`SB02MD`) and a NetSolve status variable as two additional arguments preceding the list of the parameters for the standard SB02MD routine:

```
CALL FNETSL( 'SB02MD()', STATUS, DICO, HINV, UPLO,
             SCAL, SORT, N, A, LDA, G, LDG, Q, LDQ,
             RCOND, WR, WI, S, LDS, U, LDU, IWORK,
             DWORK, LDWORK, BWORK, INFO )
```

A corresponding call from a C program is made as a function call to "`netsl`". Here the NetSolve status variable is the function return value:

```
status = netsl( "SB02MD()", &DICO, &HINV, &UPLO,
                &SCAL, &SORT, &N, A, &LDA, G, &LDG,
                Q, &LDQ, &RCOND, WR, WI, S, &LDS,
                U, &LDU, IWORK, DWORK, &LDWORK,
                BWORK, &INFO);
```

6 Concluding Remarks

The semi-automatic process described aims at minimizing and improving the tedious and error-prone work required to develop a set of different Grid interfaces for each of a large number of numerical software libraries. The underlying idea is to develop one front-end for each numerical library and one back-end for each Grid environment considered, and then with a small amount of work be able to obtain all library—Grid environment combinations. The extent to which this process can be made automatic or needs additional manual work depends both on how well structured and consistent the inline documentation is and the features provided by the Grid software. However, our experience from this proof-of-concept implementation is that a vast majority the tedious and error-prone work can better be done automatically.

The proof-of-concept implementation illustrates this process for one libary (SLICOT) and one Grid environment (NetSolve). In order to make SLICOT available in some other Grid environment or via a web portal as done in [7], the same front-end could be used with a new back-end generating, e.g., PHP-scripts for a web portal.

We remark that new numerical libraries could benefit from having the interfaces specified in, for example, XML. Based on such specification, the front-end process could be made trivial and completely automatic.

Acknowledgements

We acknowledge the anonymous referees for their comments and suggestions.

References

1. E. Anderson, Z. Bai, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide, Third Edition*. Society for Industrial and Applied Mathematics, Philadelphia, 1999.
2. P. Arbenz, W. Gander, and M. Oettli. The remote computation system. *Parallel Computing*, 23:1421–1428, 1997.
3. D.C. Arnold, H. Casanova, and J. Dongarra. Innovations of the NetSolve grid computing system. *Concurrency and Computation: Practice and Experience*, 14(13-15):1457–1479, 2002.
4. J. Bunch, J. Dongarra, C. Moler, and G.W. Stewart. *LINPACK User's Guide*. SIAM, Philadelphia, PA, 1979.
5. J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A proposal for a set of level 3 basic linear algebra subprograms. *SIGNUM Newsletter*, 22(3):2–14, February 1987.
6. J. Dongarra, J. Du Croz, S. Hammarling, and Richard J. Hanson. An extended set of Fortran basic linear algebra subroutines. *ACM Transactions on Mathematical Software*, 14(1):1–17, March 1988.
7. E. Elmroth, P. Johansson, B. Kågström, and D. Kressner. A Web Computing Environment for the SLICOT Library. In P. Van Dooren and S. Van Huffel, editors, *The Third NICONET Workshop on Numerical Control Software*, pages 53–61, 2001.
8. I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid services for distributed systems integration. *IEEE Computer*, 35(6):37–46, 2002.
9. IBM. *Engineering and Scientific Subroutine Library, Guide and Reference*. Ver. 3, Rel. 1.
10. P. Johansson and D. Kressner. Semi-Automatic Generation of Web-Based Computing Environments for Software Libraries. In *Proceedings of The 2002 International Conference on Computational Science (ICCS2002)*, 2002.
11. N. Kapaida and J. Fortes. An architecture for Web-enabled wide-area network-computing. *Journal of Networks, Software Tools and Applications*, 2(2):153–164, 1999.

12. A. Laub. A Schur method for solving algebraic Riccati equations. *IEEE Trans. Autom. Contr.*, AC-24:913–921, 1979.
13. C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, 5:308–323, 1979.
14. J. More, J. Czyzyj, and M. Mesnier. The NEOS server. *IEEE Journal on Computational Science and Engineering*, 5:68–75, 1998.
15. M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashima, and H. Takagi. Ninf: A network based information library for global world-wide computing infrastructure. In *HPCN Europe*, pages 491–502, 1997.
16. S. Shirasuna, H. Nakada, S. Matsuoka, and S. Sekiguchi. Evaluating Web services based implementations of GridRPC. In *11th IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland*. IEEE Computer Society Press, Los Alamitos, CA, 2001.
17. V. Sima. *Algorithms for Linear-Quadratic Optimization*, volume 200 of *Pure and Applied Mathematics: A Series of Monographs and Textbooks*. Marcel Dekker, Inc., New York, 1996.
18. SLICOT. The SLICOT library and the numerics in control network (NICONET) website. http://www.win.tue.nl/niconet/.
19. T. Suzumura, H. Nakada, M. Saito, S. Matsuoka, Y. Tanaka, and S. Sekiguchi. The Ninf portal: An automatic generation tool for Grid portals. In *Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande*, pages 1–7. ACM Press, 2002.
20. Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka. Ninf-G: A reference implementation of RPC based programming middleware for Grid computing. *Journal of Grid Computing*, 1(1):41–51, 2003.
21. W.M. Wonham. On a Matrix Riccati Equation of Stochastic Control. *SIAM J. Contr.*, 6:681–697, 1968.

Rapid Development of High-Performance Out-of-Core Solvers

Thierry Joffrain[1], Enrique S. Quintana-Ortí[2], and Robert A. van de Geijn[1]

[1] Department of Computer Sciences, The University of Texas at Austin, Austin, TX-78712
{joffrain,rvdg}@cs.utexas.edu
[2] Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaume I
12.071–Castellón, Spain
quintana@icc.uji.es

Abstract. In this paper, we discuss a more scalable OOC implementation of a dense linear system solver via LU factorization that presents numerical stability similar to that of the LU factorization with partial pivoting. Our implementation builds on the Formal Linear Algebra Methods Environment (FLAME), the Parallel Linear Algebra Package (PLAPACK), and the Parallel Out-of-Core Linear Algebra Package (POOCLAPACK) infrastructures. Experimental results on an Intel Itanium2 (R) platform demonstrate the high performance of this approach.

1 Introduction

Numerical linear algebra is an area that traditionally has been prone to consume vasts amounts of computer memory. When the structures that store the data are too large to fit in memory, the only solution is to rely on disk storage. Although such additional memory can be accessed via virtual memory, careful design of Out-of-Core (OOC) algorithms is generally required to attain high performance.

Practical applications of OOC computing arise in Boundary Element Methods (BEM), e.g., in the solution of integral equations in electromagnetics and acoustics. Problems with hundreds of thousands or even millions of degrees of freedom are not uncommon [4,7], leading to dense linear systems of that order.

The popular LAPACK [1] does not explicitly include OOC capabilities. For large-scale problems, prototype OOC implementations of some linear system solvers are provided in ScaLAPACK [3,5]. A more serious effort is developed in SOLAR [16]. This library uses ScaLAPACK routines for in-core computation, and also provides an I/O layer that manages matrix input-output.

As an alternative to ScaLAPACK, we developed the Parallel Linear Algebra Package (PLAPACK) [18]. Its OOC extension, POOCLAPACK, was then introduced in [13]. It is this infrastructure, together with FLAME [9,2], that we have used to implement the algorithms for dense unsymmetric linear systems described in this paper. For a complete survey on parallel OOC implementations of individual operations or machine specific libraries for dense linear systems, see [15].

In order to provide stability, OOC LU factorizations traditionally use so-called slab approaches: Blocks of columns of the matrix are brought into memory to allow the choosing of the row to be swapped from among the subdiagonal rows. These methods are inherently not scalable since, as the overall matrix problem grows, the row dimension of the slab increases and the number of columns that can fit in memory decreases so that, eventually, the cost of I/O becomes significant. (A possible manner to keep this cost bounded is described in [14].)

Thus, it is widely recognized that working with so-called (square) tiles is preferable [16,10]. The size of the tile brought into memory can always be kept constant, and therefore the ratio between the computation and I/O overhead can be fixed. The problem with this approach is that it stands in the way of partial pivoting for stability.

Our approach is different: We introduce the method of incremental pivoting to maintain most of the benefits of partial pivoting while allowing us to design an algorithm that proceeds by tiles. The result is a relatively simple, but very powerful, design. In this paper we illustrate the potential for implementing parallel versions of our approach by focusing on a sequential implementation using FLAME. Although we have a parallel implementation as well, we have not timing results yet.

The rest of the paper is organized as follows: In Section 2 we review the numerical computation of the LU factorization with partial pivoting (LUPP). In Section 3 we discuss issues regarding the LU factorization of a 2×2 blocked matrix. The insights from this study help us to present a parallel OOC algorithm for the LUPP in Section 4. Numerical stability is discussed in Section 5 and performance results are presented in Section 6. Concluding remarks are then given in the final section.

2 The LU Factorization with Partial Pivoting

Consider an $m \times n$ matrix A and its LUPP is given by

$$PA = LU, \tag{2.1}$$

where P is a permutation matrix, L is lower trapezoidal, and U is upper triangular. The LU factorization is obtained by means of a triangularization procedure also known as Gaussian elimination. Here, a sequence of permutation matrices $P_1, P_2, \ldots, P_n$ and Gauss elimination matrices $L_1, L_2, \ldots, L_n$ are computed to reduce matrix A to upper triangular form. In practice the factors L and U overwrite matrix A, and the pivots are stored in an array of n elements. The factorization of a square matrix of order n into L and U requires $2/3n^3$ floating-point arithmetic operations (or flops).

The LINPACK [6] implementation of (2.1) corresponds to an expression of the form

$$L_n^{-1} P_n \cdots L_2^{-1} P_2 L_1^{-1} P_1 A = U,$$

that does not provide the factor L explicitly. During the LINPACK algorithm the row permutations are applied to A as the matrix is factorized. The LAPACK implementation provides the lower triangular factor by rearranging the computations in this expression as

$$\hat{L}_n^{-1} \cdots \hat{L}_2^{-1} \hat{L}_1^{-1} P_n \cdots P_2 P_1 A = L^{-1} PA = U.$$

In order to do so, the LAPACK algorithm requires the row permutations to be applied to both L and A [8].

Blocked variants of these algorithms cast the bulk of the computation in terms of matrix-matrix multiplications and inherently attain high performance on modern architectures (see, e.g., [12]).

3 The LU Factorization with Incremental Pivoting

In this section, we discuss how to compute the LU factorization of the matrix

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \tag{3.2}$$

so that the LUPP of A can be reused if B, C, and/or D change. For simplicity, hereafter we consider all four quadrants in (3.2) to be of size $t \times t$, with t an exact multiple of the block size k: $t = M \cdot k$.

We recognize that we can do so with the following steps:

Step 1: Compute the LUPP

$$PA = LU.$$

The cost of this operation is $\frac{2}{3}t^3$ flops.

Step 2: Update B consistent with the factorization of A in the previous step. This is done as

$$B := L^{-1}PB,$$

at a cost of t^3 flops.

Step 3: Compute the LUPP

$$\bar{P}\begin{pmatrix} U \\ C \end{pmatrix} = \begin{pmatrix} \bar{L}_1 \\ \bar{L}_2 \end{pmatrix}\bar{U} = \bar{L}\bar{U}. \tag{3.3}$$

The key to our OOC LU factorization relies in exploiting the upper triangular structure of U in this step. By applying a blocked algorithm that proceeds k rows/columns at a time, the cost of this factorization can be reduced to $t^3 + \frac{2}{3}t^2k$. In Fig. 1 we depict graphically the three steps in the computation of an LUPP of a matrix composed of $3k$ columns ($M = 3$). The approach is fully described in the algorithm in Fig. 2. In the algorithms, the permutations are stored using a vector p. In our notation, $P(p)$ then stands for corresponding permutation matrix, of the appropriate dimensions, that is constructed from vector p. Also, $m(A)$ and $n(A)$ stand, respectively, for the number of rows and columns of matrix A; and $\{L_1 \backslash U_1\}$ denotes a matrix with L_1 and U_1 stored in its strictly lower and upper triangular parts, respectively. The algorithm produces $M = t/k$ lower triangular matrices T, of dimension $k \times k$ each, to hold $\bar{L}_1$ (see also Fig. 1) and thus avoids destroying the contents of L as computed in Step 1. It is this sequence of triangular matrices that form the diagonal blocks of $\bar{L}_1$.

As a result of this procedure, we obtain a LINPACK-like factorization of the form

$$L_M^{-1}P_M \cdots L_2^{-1}P_2L_1^{-1}P_1\begin{pmatrix} U \\ C \end{pmatrix} = \begin{pmatrix} \bar{U} \\ 0 \end{pmatrix}. \tag{3.4}$$

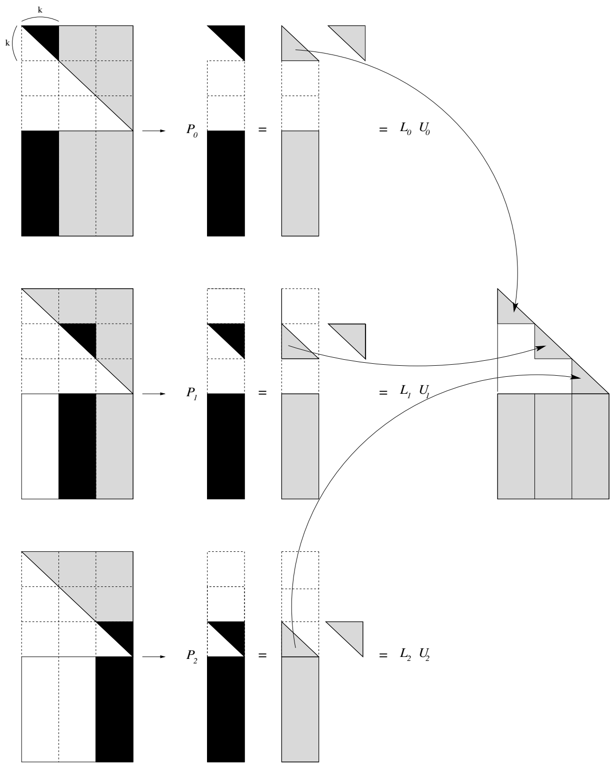


Fig. 1. LUPP factorization of $\left(U^T,\ C^T\right)^T$ exploiting the upper triangular structure of U. The LU factorization of the first, second, and third column blocks (in black) produce P_i, L_i, and U_i, $i = 0,\ 1,\ 2$, respectively. The lower triangular matrices L_i produced from each one of this factorizations together form the matrix on the right of the figure

It is important to realize that reorganizing this expression as in (3.3) destroys the structure of the lower triangular factors.

Step 4: Now, had Step 3 produced an LAPACK-like factorization as in (3.3), we would then have to apply the pivots from the previous step

$$\begin{pmatrix} B \\ D \end{pmatrix} := \bar{P} \begin{pmatrix} B \\ D \end{pmatrix},$$

and then compute the update

$$\begin{pmatrix} B \\ D \end{pmatrix} := \begin{pmatrix} \bar{L}_1^{-1} B \\ D - \bar{L}_2(\bar{L}_1^{-1} B) \end{pmatrix}.$$

Instead, Step 3 produced the LINPACK-like factorization in (3.4), and the operation that is performed in this Step is

```
Partition $U \rightarrow \left(\begin{array}{c||c} U_{TL} & U_{TR} \\ \hline\hline U_{BL} & U_{BR} \end{array}\right)$, $C \rightarrow \left(\begin{array}{c||c} C_L & C_R \end{array}\right)$, $T \rightarrow \left(\begin{array}{c} T_T \\ \hline\hline T_B \end{array}\right)$, and $p \rightarrow \left(\begin{array}{c} p_T \\ \hline\hline p_B \end{array}\right)$
  where $U_{TL}$ is $0 \times 0$, $C_L$ has 0 columns, $T_T$ has 0 rows, and $p_T$ has 0 elements

while $n(U_{BR}) \neq 0$ do
    Determine block size $k$
    Repartition
    $\left(\begin{array}{c||c} U_{TL} & U_{TR} \\ \hline\hline U_{BL} & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} U_{00} & U_{01} & U_{02} \\ \hline\hline U_{10} & U_{11} & U_{12} \\ \hline U_{20} & U_{21} & U_{22} \end{array}\right)$, $\left(\begin{array}{c||c} C_L & C_R \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} C_0 & C_1 & C_2 \end{array}\right)$,
    $\left(\begin{array}{c} T_T \\ \hline\hline T_B \end{array}\right) \rightarrow \left(\begin{array}{c} T_0 \\ \hline\hline T_1 \\ \hline T_2 \end{array}\right)$, and $\left(\begin{array}{c} p_T \\ \hline\hline p_B \end{array}\right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline\hline p_1 \\ \hline p_2 \end{array}\right)$
        where $U_{11}$ is $k \times k$, $C_1$ has $k$ columns, $T_1$ has $k$ rows, and $p_1$ has $k$ elements
    $T_1 := triu(U_{11})$   % Copy upper triangular part of $U_{11}$
    $\left[\left(\begin{array}{c} T_1 \\ \hline C_1 \end{array}\right), \ p_1\right] := \left[\left(\begin{array}{c} \{L_1 \backslash U_1\} \\ \hline L_2 \end{array}\right), \ p_1\right] = LU\left(\left(\begin{array}{c} T_1 \\ \hline C_1 \end{array}\right)\right)$   % LAPACK-like LU factorization
    $\left(\begin{array}{c} U_{12} \\ \hline C_2 \end{array}\right) := P(p_1)\left(\begin{array}{c} U_{12} \\ \hline C_2 \end{array}\right)$   % Apply permutations defined by $p_1$
    $U_{12} := L_1^{-1} U_{12}$
    $C_2 := C_2 - L_2 U_{12}$
    $triu(U_{11}) := triu(T_1)$
    Continue with
    $\left(\begin{array}{c||c} U_{TL} & U_{TR} \\ \hline\hline U_{BL} & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} U_{00} & U_{01} & U_{02} \\ \hline U_{10} & U_{11} & U_{12} \\ \hline\hline U_{20} & U_{21} & U_{22} \end{array}\right)$, $\left(\begin{array}{c||c} C_L & C_R \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} C_0 & C_1 & C_2 \end{array}\right)$,
    $\left(\begin{array}{c} T_T \\ \hline\hline T_B \end{array}\right) \leftarrow \left(\begin{array}{c} T_0 \\ \hline T_1 \\ \hline\hline T_2 \end{array}\right)$, and $\left(\begin{array}{c} p_T \\ \hline\hline p_B \end{array}\right) \leftarrow \left(\begin{array}{c} p_0 \\ \hline p_1 \\ \hline\hline p_2 \end{array}\right)$
enddo
```

Fig. 2. LINPACK-like blocked LU factorization of $\left(U^T, \ C^T\right)^T$ built upon an LAPACK-like panel factorization

$$\begin{pmatrix} B \\ D \end{pmatrix} := L_M^{-1} P_M \cdots L_2^{-1} P_2 L_1^{-1} P_1 \begin{pmatrix} B \\ D \end{pmatrix}.$$

Figure 3 presents a code for this update. It is the use of the LINPACK-like blocked LU factorization in the previous step that allows us to exploit the special structure of $\bar{L}_1$ in this step and leads to a cost of only $2t^3 + tk^2$ flops. Using a LAPACK-like blocked LU factorization destroys the structure of $\bar{L}_1$ increasing the cost of this step to $3t^3$ flops!

Step 5: Compute the LUPP

$$\hat{P}D = \hat{L}\hat{U},$$

with a cost of $\frac{2}{3}t^3$ flops.

Factoring the matrix in (3.2) using the standard LUPP costs $\frac{2}{3}(2t)^3 = \frac{16}{3}t^3$ flops. Thus, the algorithm just described only incurs additional $\frac{3}{2}t^2k + tk^2$ flops which represents a lower-order term that can be neglected if $k \ll t$.

```
Partition $B \rightarrow \left(\frac{B_T}{B_B}\right)$, $C \rightarrow \left(C_L \| C_R\right)$, $T \rightarrow \left(\frac{T_T}{T_B}\right)$, and $p \rightarrow \left(\frac{p_T}{p_B}\right)$
  where $C_L$ has 0 columns, $B_T$ and $T_T$ have 0 rows, and $p_T$ has 0 elements

while $m(B_B) \neq 0$ do
    Determine block size $k$
    Repartition
      $\left(\frac{B_T}{B_B}\right) \rightarrow \left(\frac{B_0}{\frac{B_1}{B_2}}\right)$, $\left(C_L \| C_R\right) \rightarrow \left(C_0 \| C_1 | C_2\right)$,
      $\left(\frac{T_T}{T_B}\right) \rightarrow \left(\frac{T_0}{\frac{T_1}{T_2}}\right)$, and $\left(\frac{p_T}{p_B}\right) \rightarrow \left(\frac{p_0}{\frac{p_1}{p_2}}\right)$
        where $C_1$ has $k$ columns, $B_1$ and $T_1$ have $k$ rows, and $p_1$ has $k$ elements
    $\left(\frac{B_1}{D}\right) := P(p_1)\left(\frac{B_1}{D}\right)$
    $B_1 := (tril(T_1) - diag(T_1) + I_k)^{-1} B_1$   % Solve with unit lower triangular matrix in $T_1$
    $D := D - C_1 B_1$
    Continue with
      $\left(\frac{B_T}{B_B}\right) \leftarrow \left(\frac{B_0}{\frac{B_1}{B_2}}\right)$, $\left(C_L \| C_R\right) \leftarrow \left(C_0 | C_1 \| C_2\right)$,
      $\left(\frac{T_T}{T_B}\right) \leftarrow \left(\frac{T_0}{\frac{T_1}{T_2}}\right)$, and $\left(\frac{p_T}{p_B}\right) \leftarrow \left(\frac{p_0}{\frac{p_1}{p_2}}\right)$
enddo
```

Fig. 3. Update of $\left(B^T,\ D^T\right)^T$ consistent with the LINPACK-like blocked LU factorization of $\left(U^T,\ C^T\right)^T$

4 Out-of-Core LU Factorization

In this section we show how the insights from the previous section can be used to implement a tile-based OOC LU factorization with incremental partial pivoting for matrices of arbitrary size.

Let us first examine the factorization of (3.2) where we assume that the matrix resides on disk. An OOC LU factorization can be computed as follows:

Step 1: Factor A**.** This step requires t^2 I/O operations (iops) for reading A from disk and t^2 iops more to store L, U, and the pivots back on disk.

Step 2: Update B**.** After reading B from disk, at a cost of t^2 iops, this matrix is permuted as $B := PB$, and updated by bringing panels of b columns of the lower triangular part of L in one at a time. Consider $\begin{pmatrix} 0 \\ L_{11} \\ L_{21} \end{pmatrix}$ to be one of these panels, with L_{11} lower triangular of dimension $b \times b$, and a conformal row partitioning of B as $\begin{pmatrix} B_0 \\ B_1 \\ B_2 \end{pmatrix}$. Then, we need to perform the updates $B_1 := L_{11}^{-1} B_1$ and $B_2 := B_2 - L_{21} B_1$. This requires a total of $t^2/2$ iops. Then, B is written back to disk requiring t^2 additional iops.

Step 3: Factor $\begin{pmatrix} U \\ C \end{pmatrix}$. After reading C from disk, with a cost of t^2 iops, U and C are updated by bringing panels of b rows of U into memory. Consider now the row block $(0\ U_{11}\ U_{12})$, where U_{11} is upper triangular of dimension $b \times b$, and a conformal column partitioning $(C_0\ C_1\ C_2)$ of C. Then, during this procedure, $\begin{pmatrix} U_{11} \\ \hline C_1 \end{pmatrix}$ needs to be factored and $\begin{pmatrix} U_{12} \\ \hline C_2 \end{pmatrix}$ is to be updated. This requires a total of t^2 iops (elements of U must be read and written back). Then, C is written back to disk at a cost of t^2 iops.

Step 4: Update $\begin{pmatrix} B \\ D \end{pmatrix}$. Here, D is first read from disk at a cost of t^2 iops, and then it is used in the update of B and D by bringing into memory at a time a block T_1 of dimension $b \times b$ from T, and slabs C_1 and B_1 of b rows and columns, respectively. These are used to pivot $\begin{pmatrix} B_1 \\ \hline D \end{pmatrix}$, compute $B_1 := T_1^{-1} B_1$, and finally update $D := D - C_1 B_1$. This requires $3t^2$ iops (elements of B must be read and written).

Step 5: Factor D**.** Finally, D is factored and written to disk at a cost t^2 iops.

In total, roughly $\frac{25}{2} t^2$ iops are required for the $\frac{16}{3} t^3$ *useful* flops.

Now, consider matrix A square, of order n, and partitioned into $N \times N$ square tiles of dimension t, with the (i, j) block denoted as $A_{i,j}$. It is easy to see that the above described approach can be employed to factorize the tiles in this partitioning by considering

$$\begin{pmatrix} A_{k,k} & A_{k,j} \\ A_{i,k} & A_{i,j} \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix}, \tag{4.5}$$

$k = 1, \ldots, N$, $i, j = k + 1, \ldots, N$. and applying the steps to these blocks in a certain order. In particular, our OOC LU factorization algorithm employs a blocked version of the so-called a right-looking variant of the LU factorization at the OOC level.

5 Remarks on Numerical Stability

The numerical (backward) stability of an algorithm for the LU factorization depends on the magnitude of the entries of A, L, and U during the factorization. In particular, the *growth factors* of these entries for complete and partial pivoting are bounded as $\rho^c \leq n^{1/2}(2{\cdot}3^{1/2} \cdots n^{1/n-1})$ and $\rho^p \leq 2^{n-1}$, respectively. Element growth for pairwise

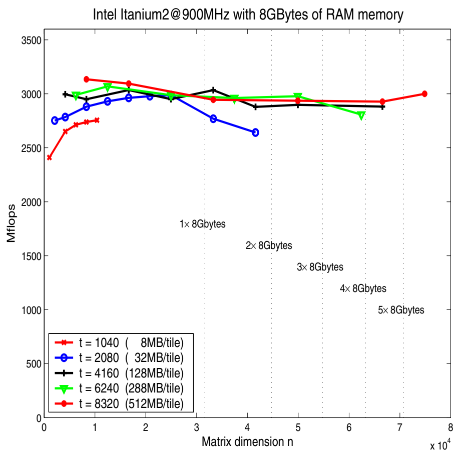


Fig. 4. Performance of the tile-based OOC LU factorization algorithm with incremental pivoting

pivoting [17] (also known as *Neville elimination*) is given by $\rho^w \leq 4^{n-1}$, respectively. This shows that neither partial nor pairwise pivoting ensure the stability of the algorithm. It is only practice that taught us to trust these approaches as, on average, the growth factors are much smaller, about $n^{1/2}$, $n^{2/3}$, and n for complete, partial, and pairwise pivoting, respectively [11].

The OOC algorithm for the LU factorization that we propose combines partial pivoting within the diagonal tiles of the matrix and a blocked version of pairwise pivoting between the diagonal block and each subdiagonal block, with partial pivoting being used again to factorize these two blocks. Thus, we can expect the numerical behavior of the OOC LU algorithm to be between those of the LU factorization with partial and pairwise pivoting. Our experiments confirm this conjecture. If necessary, iterative refinement can be employed to ensure numerical stability of pairwise pivoting (with a negligible additional cost for problems with a small number of right-hand side vectors).

For very large-scale problems, element growth should be monitored and a condition number estimator is often helpful. If element growth is unacceptable (note this can be the case using partial or pairwise pivoting), the user could rely on a similar tile-based QR factorization [10].

6 Performance

We report results for our OOC LU factorization algorithm on an Intel Itanium2 (R) (900 MHz) processor based workstation, with 8 Gbytes of RAM memory, capable of attaining 3.6 GFLOPS. High performance can be attained on this machine using only a fraction of the RAM memory. The algorithm for the (in-core) LU factorization in LAPACK delivered 3.1 GFLOPS for a square matrix of order 5200 in this platform.

In Fig. 4 we show the performance of a sequential implementation of the proposed OOC algorithm using square tiles of order t. An operation count of $2/3n^3$ for the LU factorization is used to compute the GFLOPS ratio. The results show a remarkable scalability of the OOC algorithm which is not affected by the matrix size, and an excellent performance, that rivals that of the in-core LU factorization.

7 Conclusions

We have demonstrated that a combination of the standard in-core right-looking LU factorization algorithms, together with a tile-based approach, results in a powerful new method for computing the LUPP of large, dense matrices. The tile-based approach provides true scalability and is well suited for dense linear algebra operations. The implementation of the algorithms is straight-forward, and is made particularly simple when using the FLAME environment and the PLAPACK and POOCLAPACK parallel infrastructures. Although parallel versions of the tile-based OOC LU factorization have been created using these infrastructures, these are still under evaluation.

For further information, visit `http://www.cs.utexas.edu/users/flame`.

Acknowledgments

This research was partially sponsored by NSF grants ACI-0305163 and CCF-0342369 and an equipment donation from Hewlett-Packard.

References

1. E. Anderson, Z. Bai, J. Demmel, J. E. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. E. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, 1992.
2. P. Bientinesi, J. A. Gunnels, M. E. Myers, E. S. Quintana-Ortí, and R. A. van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Trans. Math. Soft.* To appear.
3. L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, 1997.
4. Tom Cwik, Robert van de Geijn, and Jean Patterson. The application of parallel computation to integral equation models of electromagnetic scattering. *Journal of the Optical Society of America A*, 11(4):1538–1545, April 1994.
5. E. F. D'Azevedo and J. J. Dongarra. The design and implementation of the parallel out-of-core scalapack LU, QR, and Cholesky factorization routines. LAPACK Working Note 118 CS-97-247, University of Tennessee, Knoxville, Jan. 1997.
6. J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK Users' Guide*. SIAM, Philadelphia, 1979.
7. Po Geng, J. Tinsley Oden, and Robert van de Geijn. Massively parallel computation for acoustical scattering problems using boundary element methods. *Journal of Sound and Vibration*, 191(1):145–165, 1996.
8. Gene Golub and Charles Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.
9. J. A. Gunnels, F. G. Gustavson, G. M. Henry, and R. A. van de Geijn. FLAME: Formal Linear Algebra Methods Environment. *ACM Transactions on Mathematical Software*, 27(4):422–455, December 2001.

10. Brian Gunter and Robert A. van de Geijn. Parallel out-of-core computation and updating of the QR factorization. *ACM Transactions on Mathematical Software*, 2004. To appear.
11. Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.
12. B. Kågström, P. Ling, and C. Van Loan. GEMM-based level 3 BLAS: High-performance model, implementations and performance evaluation benchmark. LAPACK Working Note #107 CS-95-315, Univ. of Tennessee, Nov. 1995.
13. Wesley C. Reiley and Robert A. van de Geijn. POOCLAPACK: Parallel Out-of-Core Linear Algebra Package. Technical Report CS-TR-99-33, Department of Computer Sciences, The University of Texas at Austin, Nov. 1999.
14. S. Toledo. Locality of reference in LU decomposition with partial pivoting. *SIAM J. Matrix Anal. and Appl.*, 18(4):1065–1081, 1997.
15. Sivan Toledo. A survey of out-of-core algorithms in numerical linear algebra. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1999.
16. Sivan Toledo and Fred G. Gustavson. The design and implementation of SOLAR, a portable library for scalable out-of-core linear algebra computation. In *Proceedings of IOPADS '96*, 1996.
17. L. N. Trefethen and R. S. Schreiber. Average-case stability of Gaussian elimination. *SIAM J. Matrix Anal. and Appl.*, 11(3):335–360, 1990.
18. Robert A. van de Geijn. *Using PLAPACK: Parallel Linear Algebra Package*. The MIT Press, 1997.

ALPS: A Software Framework for Parallel Space-Time Adaptive Processing

Kyusoon Lee and Adam W. Bojańczyk

School of Electrical and Computer Engineering
Cornell University
Ithaca, NY, 14850
{kl224,awb8}@cornell.edu

Abstract. Space-Time Adaptive Processing (STAP) refers to adaptive radar processing algorithms that take the signals from both multiple sensors and multiple pulses to cancel interferences and detect a target. Fully-adaptive STAP is known to be optimal, but the required number of operations is overwhelming, and makes this method impractical. Hence, many different heuristic approaches are sought to approximate the optimal method with smaller number of operations. In this work, we present a software framework called *ALPS* to help prototype various parallel STAP methods, and predict their performances.

1 Introduction

Space-Time Adaptive Processing (STAP) refers to a class of adaptive radar processing algorithms. In space-time adaptive processing, a series of M pulses are collected from N sensors multiple (L) times. Hence, the data is viewed a three-dimensional cube of size $L \times M \times N$ as in Figure 1. Typical dimensions are: $16 \leq M \leq 256$, $16 \leq N \leq 256$, and $L \approx MN$. *Fully-adaptive* STAP algorithm solves a constrained minimization problem defined on this cube to calculate a quantity known as a *weight vector*. This algorithm require $O(L^2MN) \approx O(M^3N^3) \approx 10^9$–$10^{12}$ operations [1], which cannot be delivered in real-time with current technology.

Hence, *partially-adaptive* techniques which approximate the fully-adaptive algorithm and require fewer number of operations have been studied [1]. In these algorithms, the data cube is transformed into smaller size sub-cubes which are next subjected to

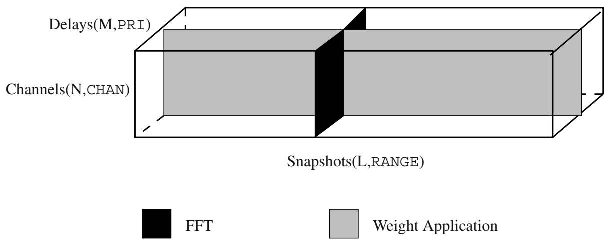


Fig. 1. Three-dimensional data cube

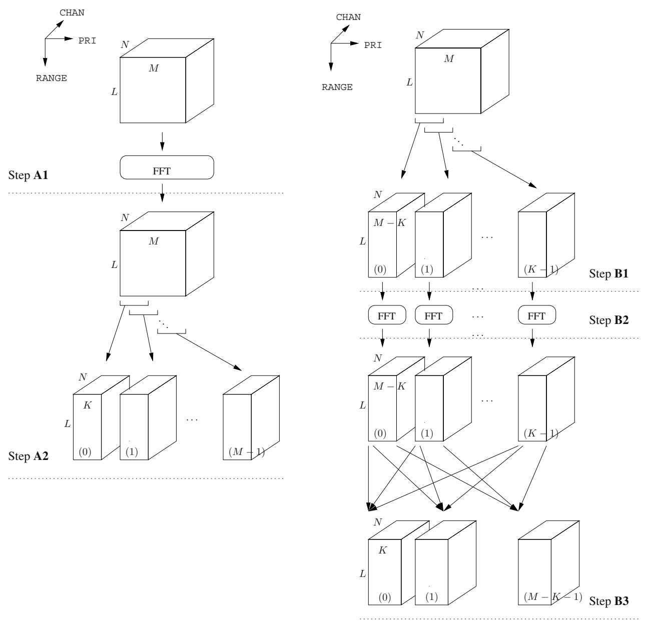


Fig. 2. Two representative pre-STAP processings (a) PRI-overlapped (left), (b) PRI-staggered (right)

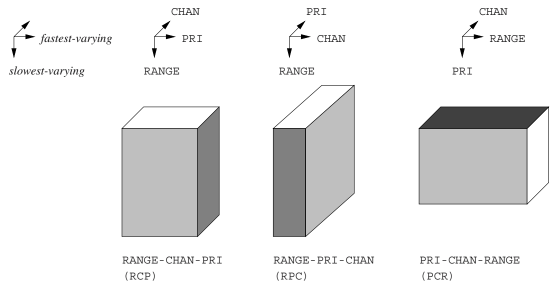


Fig. 3. Different orientations

adaptive processing. This downsizing transformation is called *pre-STAP processing*. Two representative approaches are *PRI-overlapped* and *PRI-staggered* methods as illustrated in Figure 2. In these methods, STAP algorithms can run on many sub-cubes simultaneously. Computed weight vectors on these sub-cubes are combined together to give an approximation to a fully-adaptive weight vector. Because of the parallelism and smaller sizes of sub-cubes, partially-adaptive algorithms can be run in real time.

However, the optimal parallel implementation of these algorithms is far from obvious. For example, FFT operations in pre-STAP processing and weight application in STAP require access to orthogonal dimensions of data cube as in Figure 1. This suggests that different memory layout of a data cube, or its different distribution among processors for different operations might lead to different performances. We call the mapping from three physical dimensions (`PRI`, `CHAN`, and `RANGE`) of a data cube to fastest-, intermediate-, slowest-varying dimension of a memory an *orientation*. Figure 3 shows three different orientations of a data cube that we are currently considering.

Parallel processing of STAP algorithms has been considered by many researchers. Basic theories and techniques for parallel pipelined implementation of STAP algorithms were presented in [13], [14]. Software systems for designing portable, optimized parallel signal processing algorithms based on PVL and S3P were developed by MIT Lincoln Laboratory [15], [16].

In [17], we introduced another framework for prototyping STAP methods that also considers the three-dimensional *topology* of processors that allows users to easily describe various STAP algorithms, to build execution time models of the basic building blocks, and to optimize the algorithms based on these models automatically. In that framework, we considered the three-dimensional *topology* of processors and the *orientation* of data cubes as variables in the performance optimization process while others considered only the *number* of processors as a variable and left other decisions to users [15], [16]. This work presents modification to a modeler described in [17]. We ported the ALPS software framework onto linux machines, and here we present the results from a linux cluster as well as the one from a windows cluster. We also extended this software framework to answer questions such as, given the time constraint, how large data cubes can be processed.

2 Algorithmic Library for Parallel STAP (ALPS)

In various STAP methods, similar tasks are processed in different order [4]. For example, the tasks in step **A1** and step **B2** in Figure 2, are both FFT operations. The tasks of the data cube reorganization in steps **A2** and **B1**, which we call `split-staggered` operation, are also the same. It turns out that various partially-adaptive STAP algorithms can be implemented with a small number of such tasks [4]. Based on this observation, we built a parallel library called *ALPS* which includes implementations of common STAP tasks.

In the ALPS library, a data cube is modeled as a three-dimensional data located on a three-dimensional processor cube with a block-cyclic distribution. ALPS library provides the transformation routines (`split-staggered`, `join`, `recube`) to cover various pre-STAP processings such as those shown in Figure 2. Complete space-time adaptive processing routines (`fullupdate`, `predictres`) are included too. It also provides communication routines to change the processor cube dimensions (`fewtomany`, `manytofew`) and to change the orientation of 3-dimensional data cube (`transpose`).

Figure 4 shows the task graph of a prototype partially-adaptive STAP algorithm with PRI-staggered pre-STAP processing (called *PRI-staggered STAP*) implemented with ALPS library routines. In the figure, `fullUpdate` STAP methods operate on three data cubes; *training cube* to compute a weight vector, *steer cube* which specifies the constraints, and *search cube* which is filtered according to the direction of the computed weight vector. All these cubes undergo the same pre-STAP processing. `join` at the end of each algorithm merges the results from `fullUpdate` operations. Figure 5 shows a

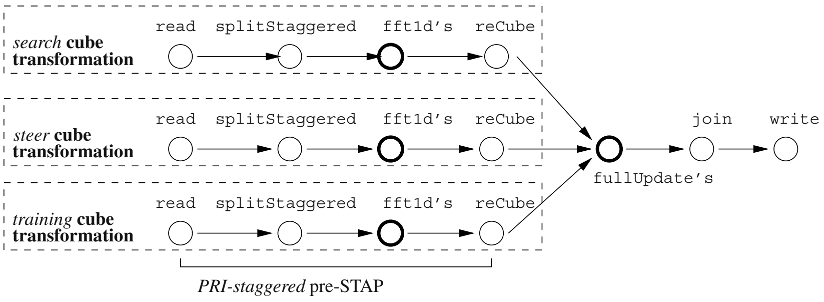

search cube transformation
steer cube transformation
training cube transformation
read
splitStaggered
fft1d's
reCube
fullUpdate's
join
write
PRI-staggered pre-STAP

Fig. 4. Task graph of PRI-staggered STAP described with ALPS building blocks

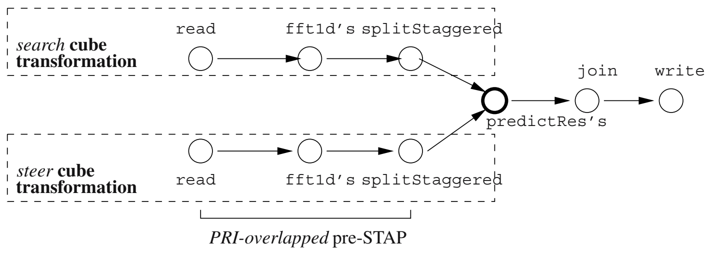


Fig. 5. Task graph of PRI-overlapped STAP described with ALPS building blocks

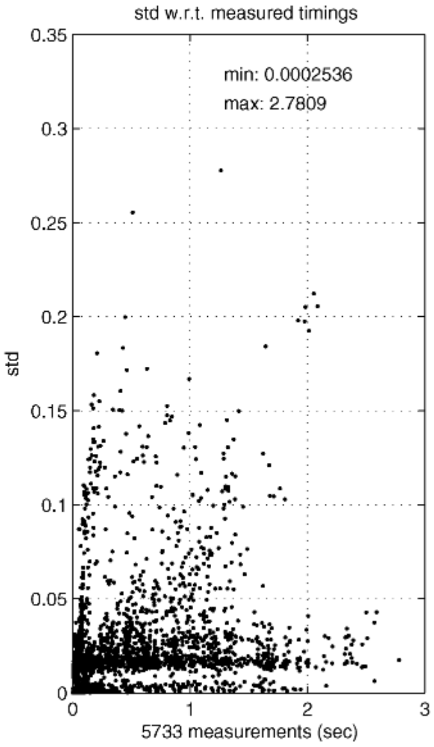
similar task graph of a prototype partially-adaptive STAP algorithm with PRI-overlapped pre-STAP processing (called *PRI-overlapped STAP*)

3 ALPS Benchmarking and Modeling

We assume that the execution time can be described as a function of input parameters. For example, `fewtomany` operation moves a data cube of size (d_x, d_y, d_z) on a processor cube of size (p_x, p_y, p_z) to a processor cube of size (p'_x, p'_y, p'_z). Hence, the execution time model of `fewtomany` is assumed to be a function of these parameters. `ALPS Benchmarker` runs each routine and records its execution time for given values of input parameters. The range for the input parameters is specified by the user. `ALPS Modeler` constructs execution time models by solving multiple linear regression problems for these parameters, and reports many metrics including the mean of relative errors of a model:

$$\text{(mean of relative errors, m.r.e.)} \equiv \frac{1}{\#\text{cases}} \sum \frac{|\text{prediction}_i - \text{measurements}_i|}{\text{measurements}_i}$$

In [17], we observed rather high mean of relative errors for communication routines. To understand where this large error results from, we drew a scattered plot of mean of 5 replications for each measurement versus the standard deviation, and the standard deviation over the mean as in Figure 6. A commercial off-the-shelf (COTS) linux cluster



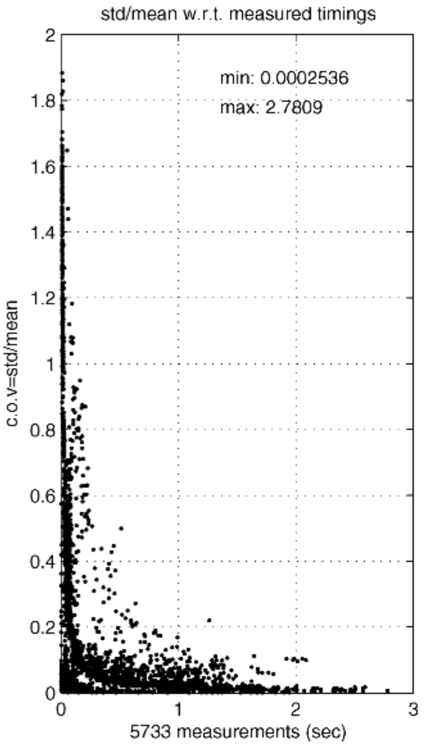


Fig. 6. Scattered plot of measurement vs. standard deviation (left) and coefficient of variance = standard deviation / mean (right) of 5 replications

of 8 PCs with a single switch network was used to plot Figure 6. Another experiment on a COTS windows cluster with 64 PCs with a gigabit network showed similar patterns. The relative standard deviation defined as the ratio of the standard deviation and the mean, can be as large as 200% of the mean for measurements shorter than 0.1 sec. These large relative standard deviations are the causes of large relative error in the model.

One way of improving the model is to eliminate outliers. In one experiment, we removed those measurements with relative standard deviations larger than 50%. Then, we randomly picked 70% of measurements to build execution time models, and used the other 30% to validate these models. The results are shown in Figure 7. The mean of relative errors for the communication routines (`FewToMany`, `ManyToFew`, and `Transpose`) is around 50%. Eliminating those outliers made the mean of relative errors on the test set comparable to the mean of relative errors on the training set. In [17], the mean of relative errors on the test set was always substantially higher than the mean of relative errors on the training set.

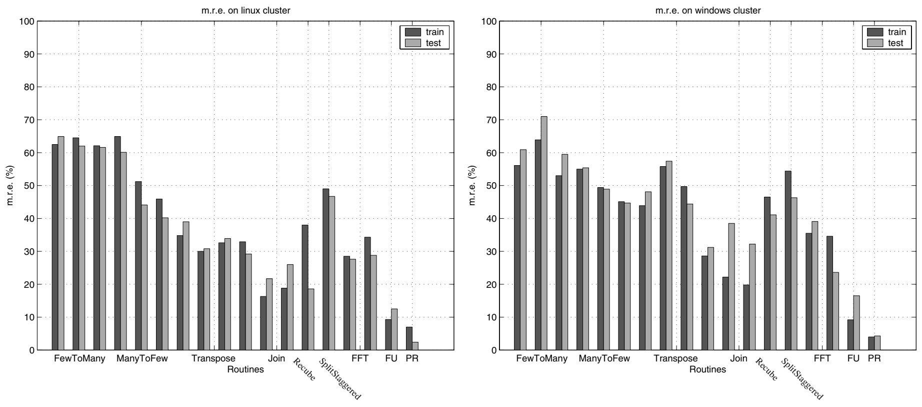


Fig. 7. Mean of relative errors (m.r.e.) on training set and test set

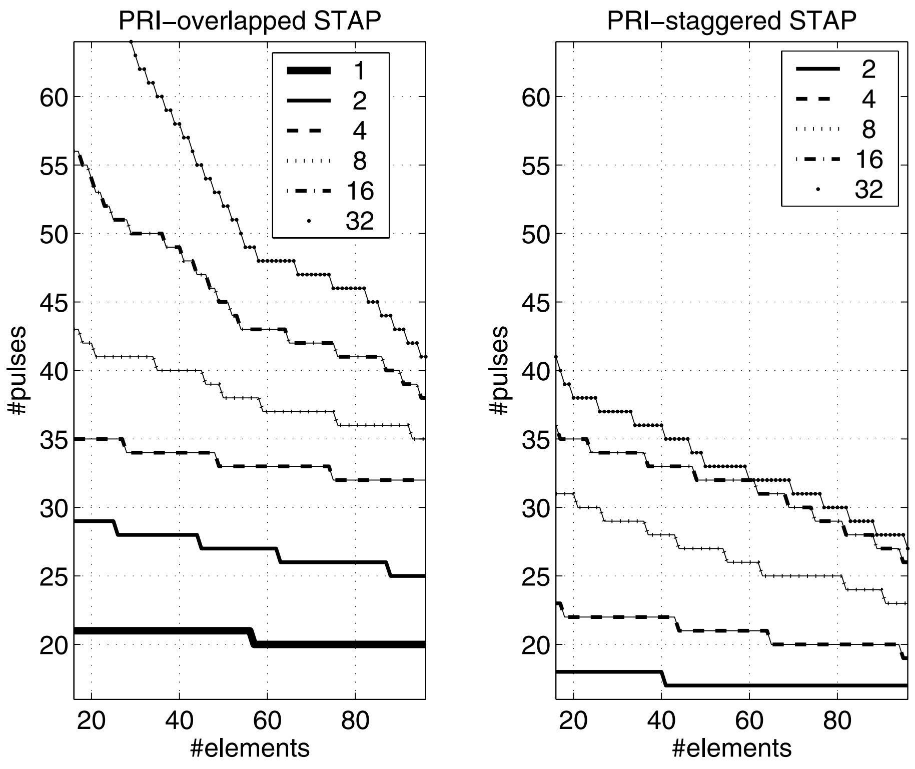


Fig. 8. PRI-overlapped vs. PRI-staggered STAP: 2048 snapshots, deadline 100 sec

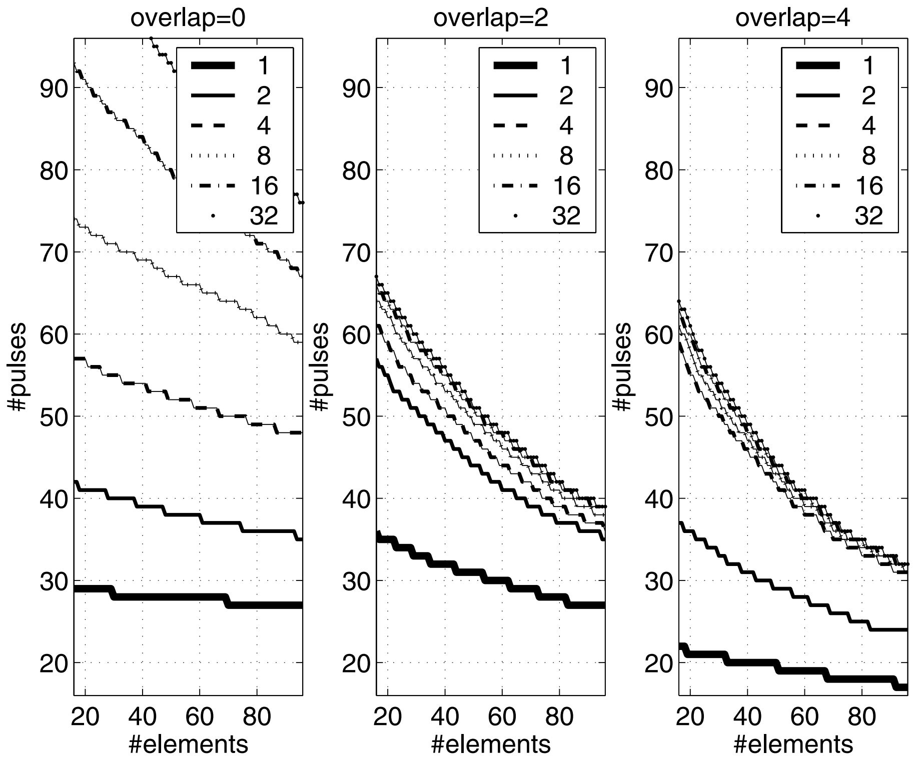


Fig. 9. PRI-overlapped vs. overlap parameter: 2048 snapshots, deadline 20 sec

4 STAP Algorithm Performance Optimization and Prediction

While general processor assignment problems on a task graph such as Figure 4 are NP-complete [9], [11], those problems on a linear task graph or a layered structure can be solved by dynamic programming [12].

Dynamic programming is a technique to find a minimum-cost path on a layered problem. With some modification, it can be applied to our problem of deciding the optimal processor cube dimension and orientation for tasks in a task graph like the one in Figure 4. There, we have a node with multiple incoming paths. For such nodes, we redefined the recurrence function of dynamic programming to be the sum of minimum costs from all preceding tasks. This means that any tasks in one path cannot be run simultaneously with tasks in another path.

In [17], we introduced a software tool called `ALPS Optimizer` to perform the modified dynamic programming optimization for task graphs such as that in Figure 4 and 5. `ALPS Optimizer` accepts the pseudo-codes corresponding to these task graphs and finds the optimal orientations and processor cube dimensions and the minimum time that it takes to run these task graphs.

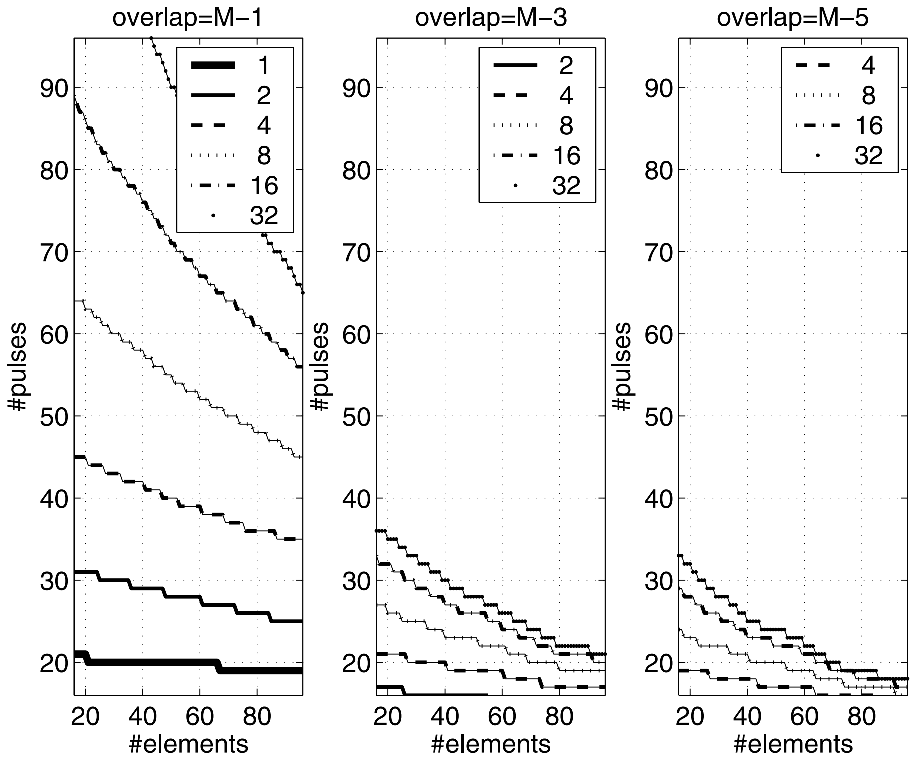


Fig. 10. PRI-staggered vs. overlap parameter: 2048 snapshots, deadline 20 sec

Based on the best execution time returned by this `ALPS Optimizer`, we could compare, given the execution time constraint and the number of available processors, how large data cubes can be processed by different partially-adaptive STAP algorithms. Figure 8 shows the number of pulses(`PRI` dimension), and the number of elements(`CHAN` dimension) that each partially-adaptive STAP algorithm can process, given the time limit of 100 sec, and the resource limit of $1, 2, 4, 8, 16$, and 32 processors. It is worth noting that with the algorithmic parameters that we assumed here, PRI-staggered STAP is very computationally demanding, hence could not meet the time limit of 100 sec when run on one processor. On the other hand, PRI-overlapped was able to process data within given time limit. We also could see how large the number of pulses and the number of elements can be processed within a given time limit when the algorithmic parameter `overlap` changes. In Figure 9 and 10, the values of `overlap` are varied. The plots show dimensions of data cubes which can be processed within the set deadline of 20 sec. In this way, users can see the trade-off between the change in the algorithmic parameter, and the corresponding execution time.

5 Conclusion

Space-time adaptive processing techniques must be implemented on parallel computers if they are to meet real-time processing requirements. However, the complexity of these algorithms and the difficulty of deciding optimal processor cube dimension and orientation for each task makes it hard to find the optimal parallel implementation by hand.

In this work, we extended the ALPS software framework that was first introduced in [17] for prototyping parallel STAP methods. We investigated where the large modeling errors came from, and modified the execution time modeler accordingly. We also extended its optimizing tools to answer questions such as the size of data cubes that various STAP algorithms can process within certain time limit and resource limit. Examples illustrated our general approach, and showed that this approach is feasible.

References

1. James Ward. Space-time adaptive processing for airborne radar. Technical report 1014, Massachusetts Institute of Technology Lincoln Laboratory, Lexington, MA, December 1994.
2. R.C. DiPietro. Extended factored space-time processing for airborne radar systems. In *Twenty-sixth Annual Asilomar Conference on Signals, Systems, and Computing*, pages 425–430, Pacific Grove, CA, 1992.
3. L.E. Brennan and F.M. Staudaher. Subclutter Visibility Demonstration. Technical Report RL-TR-92-21, Adaptive Sensors Incorporated, March, 1992.
4. James M. Lebak and Adam W. Bojanczyk, Design and Performance Evaluation of a Portable Parallel Library for Space-Time Adaptive Processing. IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 3, March 2000.
5. R. Dimitrov and A. Skjellum, An efficient MPI implementation for Virtual Interface (VI) Architecture, September 11th, 1999.
http://www.mpi-softtech.com/company/publications/?view=1037051037
6. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, third edition, 1999.
7. Frigo, Matteo and Johnson, Steven G. FFTW: An adaptive software architecture for the FFT. Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing, pages 1381–1384, Vol. 3, 1998.
8. Jame M. Lebak, Robert C. Durie, and Adam W. Bojanczyk. Toward A Portable Parallel Library for Space-Time Adaptive Methods. Technical Report CTC96TR242, Cornell University, June, 1996.
9. Hesham El-Rewini, Theodore G. Lewis and Hesham H. Ali. *Task Scheduling in Parallel and Distributed Systems*, Prentice Hall, 1994.
10. Andreas S. Schulz. Scheduling to Minimize Total Weighted Completion Time: Performance Guarantees of LP-Based Heuristics and Lower Bounds. IPCO: 5th Integer Programming and Combinatorial Optimization Conference, 1996, pages 301–315.
11. Anant Singh Jain and Sheik Meeran. A State-of-the-Art Review of Job-Shop Scheduling Techniques, Technical Report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.
12. Frederick S. Hillier and Gerald J. Lieberman. *Introduction to Mathematical Programming*. McGraw-Hill, second edition, 1995.

13. Alok Choudhary, Wei-keng Liao, Donald Weiner, Pramod Varshney, Richard Linderman and Mark Linderman. Design, Implementation and Evalutation of Parallel Pipelined STAP on Parallel Computers, 12th. International Parallel Processing Symposium, March 30–April 03, 1998, pages 220–225.
14. Myungho Lee and Viktor K. Prasanna. High Throughput-Rate Parallel Algorithms for Space Time Adaptive Processing, 2nd International Workshop on Embedded Systems and Applications, Apr. 1997.
15. Edward Rutledge and Jeremy Kepner. PVL: An Object Oriented Software Library for Parallel Signal Processing, IEEE International Conference on Cluster Computing, Oct 8–11, 2001.
16. Jeremy Kepner, DoD Sensor Processing: Applications and Supporting Software Technology, Supercomputing 2002 Tutorial S14, Nov. 17, 2002.
17. Kyusoon Lee and Adam W. Bojanczyk, Performance Modeling and Optimization Framework for Space-Time Adaptive Processing (STAP), 3rd International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS' 2004), April 26–April 30, Santa Fe, New Mexico.

Hybrid Parallelization of CFD Applications with Dynamic Thread Balancing

Alexander Spiegel[1], Dieter an Mey[1], and Christian Bischof[1,2]

[1] Center for Computing and Communications, RWTH Aachen University, Germany
{spiegel,anmey,bischof}@rz.rwth-aachen.de
[2] Institute for Scientific Computing and Center for Computing and Communications
RWTH Aachen University, Germany

Abstract. SMP Clusters with fat nodes offer an interesting capability for large applications that employ a hybrid parallelization model: to improve load balance, the number of threads can be increased in order to speed-up busy MPI processes or decreased to slow down idle MPI processes, provided these processes reside on the same SMP node. We developed a library which performs this thread adjustment automatically during program execution. Experimental results demonstrate remarkable speed-ups with minimal programming effort.

1 Introduction

Hybrid parallelization using message-passing and multi-threading with OpenMP or autoparallelization on clusters of shared-memory computers is not always profitable [1,3], yet it offers interesting opportunities [2], particularly on SMP clusters with fat nodes. If an application employs a hybrid parallel programming model utilizing both MPI and threads (from now on we call such an application a "hybrid application" for simplicity), the number of threads can be increased in order to speed-up busy MPI processes or decreased to slow down idle MPI processes, provided these processes reside on the same SMP node.

It may not always be easy to find out the optimal distribution of threads to the MPI processes, however, and the optimal distribution may change in the course of the runtime of an application. Therefore, we developed a dynamic thread balancing (DTB) library which performs this thread adjustment automatically, requiring the user just to insert one extra MPI call in the code to trigger this feature.

The PMPI profiling interface, which is part of the MPI specification and therefore part of each compliant implementation, offers an easy and portable way of intercepting calls to the MPI library. The DTB library uses this interface to capture the user time of each MPI process and the time spent in MPI routines. This information is evaluated to adjust the number of threads for each MPI process, taking care not to oversubscribe the number of processors dedicated to the application. The major features and the implementation of the DTB library are described in section 2.

Experiments were performed on a Sun Fire SMP cluster with the new multi-zone version of the BT code of the NAS Parallel Benchmark suite and the FLOWer Navier-Stokes solver and the results are reported on in section 3. Section 4 concludes with a summary and an outlook on future work.

2 The Dynamic Thread Balancing (DTB) Library

2.1 Usage of the DTB Library

The DTB library has to be linked between the user program and the MPI library. One or more calls to the newly provided routine `MPI_Pcontrol` have to be inserted in suitable spots of the program code in order to regularly - typically once per outer loop iteration (e.g. a time step in a CFD application) - trigger the steering mechanism. Thus, programming effort for the user of the DTB library is minimal. The calls of the `MPI_Pcontrol` routine are ignored when the DTB library is not linked. The user time (time spent in user routines) and the MPI overhead (time spent in MPI routines) between two successive `MPI_Pcontrol` calls of each MPI process are measured and evaluated to adjust the number of threads. The DTB steering mechanism shifts threads between the MPI processes residing on the same node with the OpenMP `omp_set_num_threads` call. If freed threads stay in a busy waiting state, they still consume compute cycles. Therefore they have to be put to sleep completely. On Solaris we hence set the environment variable `SUNW_MP_THR_IDLE` to `sleep`. Of course the more MPI processes reside on one node and the more CPUs are available on a node, the better the algorithm is able to shift threads between the MPI processes and to improve load balancing.

When using multiple SMP nodes, the initial distribution of MPI processes onto the nodes is very important: If all the busiest processes are started on the same node, the DTB algorithm has little to gain. It is thus advantageous to first execute a shorter profiling run - with the balancing mechanism of the DTB library turned off - and then use an optimized distribution for the production run.

2.2 Prediction of the Runtime of the MPI Process

In each iteration the number of threads and the user time is stored for each MPI process. This information is then used to calculate an approximation function $T(t)$ of the runtime of each MPI process depending on the number of threads t

$$T(t) = T_s + T_p/t + T_o * t$$

with

$$\begin{aligned} T_s &= serial\ part\ of\ the\ runtime \\ T_p &= parallel\ part\ of\ the\ runtime \\ T_o &= parallelization\ overhead \end{aligned}$$

by using the least square method to determine T_s, T_p and T_o.

If the program behaves smoothly, the corresponding linear equation system will have a unique solution with non-negative components. If the system does not have a unique solution, which typically happens during the start phase or if the thread distribution process has well stabilized, the approximation formula is simplified by setting T_o to zero or T_o and T_s to zero. If the solution has negative components, for example because a loaded system disturbs the program performance, the optimal positive solution can be determined by setting one or even two out of the three parameters to zero. In any case, in an overloaded system any increase of the number of threads will diminish the performance and the DTB library will therefore most likely reduce the thread number.

In order to adapt to a changing runtime behavior, an aging factor is employed to give more recent timing measurements a higher weight. In each iteration the impact of all previous measurements is reduced depending on the difference between the recently predicted runtime and the currently measured runtime. This aging factor can also be fixed by an environment variable.

At program start we can activate a warm-up phase in order to speed-up the initial balancing process. Thereby we quickly gather three values of $T(t)$ for the approximation formula by explicitly varying the number of threads t. This helps to avoid linear equation systems without a unique solution in the beginning. If warm-up is disabled, all processes are started with only one thread and all remaining threads are put into the thread pool (see below).

2.3 The DTB Steering Algorithm

A pool of unused threads is managed by the following inexpensive steering mechanism:

- Search for the MPI process taking maximum compute time (`maxwork_task`). (This process has to be accelerated.)
- If the thread pool is empty, search for the MPI process with more than one thread and minimum predicted compute time with one less threads (`minwork_task`). (This process will suffer the least from losing one thread.)
- If the thread pool is empty and the predicted compute time of `minwork_task` with one less thread is less than the current compute time of `maxwork_task`, reduce the number of threads of process `minwork_task` by one and increase the thread pool. (This process may loose one thread, without slowing down the whole application.)
- If the thread pool is not empty and if the predicted compute time of process `maxwork_task` with one more thread is less than the current maximum compute time, increase the number of threads of process `maxwork_task` and reduce the thread pool by one. (The busiest process really profits from an additional thread.)
- Else if the predicted compute time of process `maxwork_task` when loosing one thread is less than the current maximum compute time, decrease the number of threads of process `maxwork_task` and increase the thread pool by one. (The busiest process does not scale well. In fact, one less thread might be profitable.)

2.4 Limitations of the DTB Library

The DTB library currently is limited to the masteronly hybrid programming method (described, for example, in [3]), in which only the master thread calls the MPI library routines outside of any parallel regions. As DTB uses the PMPI interface, other MPI profiling tools like Vampirtrace can not be combined with the DTB library. Lastly, the current thread assignment mechanism does not yet take the data affinity of threads into account. Thus, we may experience some minor performance degradation on a ccNUMA machine when threads are shifted between MPI processes.

3 Experimental Results

3.1 Computing Environment

Timing experiments were performed on the Sun Fire SMP Cluster at RWTH Aachen University. It consists of two groups of eight Sun Fire 6800 nodes each and one group of four Sun Fire 15K nodes which are tightly connected by a fast Sun Fire Link network. The Sun Fire 6800 nodes are equipped with 24 UltraSPARC III Cu processors running at 900 MHz and the Sun Fire 15K nodes contain 72 processors of the same kind. The current production environment is Solaris 9 8/03 and we use the Sun ONE Studio 8 compilers. The Sun HPC ClusterTools 5 package includes a fully thread-safe MPI 2 implementation.

The DTB library also uses MPI for internal communication itself and has to do some minor computations. In our experiments, we observed an overhead of up to 5 percent for the library, but usually this is more than made up for by the resulting runtime improvements.

3.2 NAS Parallel Benchmark BT - Multi-zone Version (Class A)

We used the BT benchmark from the NAS Parallel Benchmark collection (NPB) (see [7,8,9]) for testing the DTB library. The BT benchmark solves discretized versions of the unsteady and compressible Navier-Stokes equations in three spatial dimensions. In order to profit from multi-level parallelism recently a multi-zone (MZ) version of the benchmark has been developed.

The BT-MZ benchmark is a good experimental choice, because the overall mesh is partitioned such that the size of successive zones in one particular coordinate direction is increased in a roughly geometric fashion. The class A benchmark contains 16 zones and the ratio of the largest over the smallest total zone size is approximately 20. As a result, this program suffers from an inherent load imbalance. To illustrate, tables 1 and 2 show the timing distribution among 8 and 16 MPI processes on a Sun Fire 15K. In the 8-processor run, the user time of the process with rank 0 is about 2 times higher than the user time of the processes with ranks 6 and 7. With 16 processors, the user time of the process with rank 0 is now over 22 times higher than the user time of the process with rank 15. In fact, the latter process spends only about 4 percent of its overall runtime doing useful work.

The BT-MZ code features hybrid parallelism with OpenMP on the lower level. The BT-MZ developers also implemented a balancing mechanism which determines a fixed number of threads for each process based on the grid size of the zones at program start. We compare the performance of the unmodified BT-MZ code with the built-in balancing turned off and on to our DTB mechanism in tables 3 and 4. The activation of the DTB load balancing required us to add two invocations of the `MPI_Pcontrol` routine.

These tables show the overall runtime (that is, user time plus MPI Overhead) versus the numbers of processors (which equals the sum of the number of OpenMP threads of all MPI processes). For 8 (16) MPI processes, the hybrid parallelism built into BT-MZ can employ additional processors in multiples of 8 (16), hence there are no timing numbers for a number of processors that does not fulfill this criterion. The DTB library,

Table 1. NPB BT-MZ Class A benchmark with 8 MPI-Processes on one Sun Fire 15K; runtime of the single MPI processes in seconds

Rank-ID	User Time	MPI Overhead	Rank-ID	User Time	MPI Overhead
0	92.8	0.5	4	46.5	46.9
1	56.7	36.6	5	42.8	50.6
2	56.0	37.3	6	44.2	49.1
3	44.8	48.5	7	44.5	48.8

Table 2. NPB BT-MZ Class A benchmark with 16 MPI-Processes on one Sun Fire 15K; runtime of the single MPI processes in seconds

Rank-ID	User Time	MPI Overhead	Rank-ID	User Time	MPI Overhead
0	90.4	0.5	8	18.7	72.0
1	55.1	35.8	9	18.9	72.0
2	54.9	36.0	10	11.6	79.3
3	33.7	57.2	11	11.5	79.4
4	31.0	59.9	12	10.9	78.0
5	31.2	59.7	13	6.7	84.2
6	19.1	71.8	14	6.7	84.3
7	19.1	71.8	15	4.0	86.9

Table 3. NPB BT-MZ Class A benchmark with 8 MPI-Processes on one Sun Fire 15K; comparison of runtime with different thread distribution methods in seconds

#procs	built-in balanc. off	built-in balanc. on	DTB	#procs	built-in balanc. off	built-in balanc. on	DTB
8	92.1			22			29.0
10			55.7	24	40.8	34.8	35.1
12			46.4	26			28.7
14			46.5	28			27.2
16	66.6	43.7	44.7	30			24.3
18			38.0	32	31.9	25.6	24.1
20			34.0				

on the other hand, does not have such a restriction, and can employ an arbitrary number of threads.

The results in tables 3 and 4 show the remarkable impact of DTB on the performance of BT-MZ. While the runtimes are not monotonically decreasing, due to interference with other unrelated jobs running on the 72-processor Sun Fire 15K, overall the use of DTB results in considerable improvement. In particular, just adding 2 additional threads results in a roughly 40 percent improvement in runtime for both 8 and 16 MPI processes, as the

Table 4. NPB BT-MZ Class A benchmark with 16 MPI-Processes on one Sun Fire 15K; comparison of runtime with different thread distribution methods in seconds

#procs	built-in balanc. off	built-in balanc. on	DTB	#procs	built-in balanc. off	built-in balanc. on	DTB
16	89.6			26			31.6
18			55.1	28			26.1
20			41.8	30			25.6
22			35.4	32	60.2	24.8	24.5
24			32.4				

Table 5. NPB BT-MZ Class A benchmark with 16 MPI-Processes on two Sun Fire 6800; comparison of runtime with different thread distribution methods in seconds

#threads per MPI process	#processors	built-in balancing off	DTB worst case	DTB best case
1	16	89.5		
2	32	47.7	31.2	23.4
3	48	37.6	27.1	21.5

DTB library is able to allocate these additional resources in a fashion that substantially reduces load imbalance. Further it is noteworthy that the runtimes of the hand-coded, BT-specific load balancing scheme and of the much less programming-intensive DTB parallelization are comparable.

Lastly, in table 5 we describe an experiment with 16 MPI processes distributed over two Sun Fire 6800 nodes. In this case, the BT-specific load balancing mechanism cannot be employed as it assumes that all processes have access to the same shared memory. In such a case, the initial distribution of processes to nodes has an important impact on how much can be achieved with the DTB mechanism. Starting the first 8 processes on the first node and the second 8 processes on the second node is the worst case, as most of the work resides on the first node, whereas distributing the processes alternatingly to the nodes is the best case.

We conclude that in all experiments the DTB load balancing approach worked well. In cases where the built-in balancing scheme based on the grid sizes is applicable, it performs quite similar to the DTB mechanism. However, the DTB library offers greater flexibility with respect to the number of CPUs that can profitably be employed, and in particular already adding just a few processors results in greatly reducing load imbalance. Given that we had to add just two lines of code to the "unbalanced" BZ-MZ benchmark, the DTB library offers a very good return for a very limited programming investment.

3.3 The FLOWer Navier-Stokes-Solver

In an ongoing project sponsored by the German Research Council (DFG), scientists of the Laboratory of Mechanics of RWTH Aachen University are simulating PHOENIX, a small scale prototype of the Space Hopper, a space launch vehicle designed to take

Table 6. FLOWer on one Sun Fire 6800; 6 processes with 2 threads each; runtime of the single MPI processes in seconds

Rank-ID	User Time	MPI Overhead	Rank-ID	User Time	MPI Overhead
0	258.5	253.3	3	187.5	324.3
1	182.8	329.0	4	186.0	325.8
2	210.6	301.2	5	497.6	14.2

Table 7. FLOWer on one Sun Fire 6800; runtime with and without DTB in seconds

#processors	#MPI processes	#threads	without DTB	with DTB
12	6	2	499.8	350.6
14	7	2	606.9	450.7
16	8	2	511.0	317.0
18	6	3	408.8	286.3
21	7	3	474.3	347.1
24	6	4	363.2	314.0

off horizontally and glide back to earth after placing its cargo in orbit (see [4,5,6]). The corresponding Navier-Stokes Equations are solved on a block structured grid with FLOWer, a flow solver developed at the German Aerospace Center (DLR).

The code is parallelized with the CLIC-3D communication library which encapsulates all the MPI communication. As all information exchange is grid block-oriented, the number of grid blocks - which in our case is 20 - limits the number of MPI processes. When less MPI processes are started than blocks are used, FLOWer distributes the blocks over the processes according to the grid sizes in an effort to balance computational load. Underneath the coarse-grained parallelization with CLIC-3D/MPI, the most computationally expensive loop nests can be efficiently parallelized using threads.

We carried out similar experiments as with the BT-MZ benchmark on a 24-processor Sun Fire 6800. In table 6, the runtimes of a job with six MPI processes, each using 2 threads, is shown in detail. It can be seen that the compute time of the various processes differs by a factor of over two.

In table 7, we show the impact of adding more threads, with or without employing the DTB library. We noticed that an increase of the number of MPI processes not necessarily improves performance, which probably has to do with the fact that a greater number of MPI processes leads to a greater communication overhead overall.

However, more importantly, the use of the DTB library resulted in substantial improvements in any case. Given that FLOWer is a code used in production in many projects, these results show the benefit of hybrid parallel programming and the performance potential that can be gained from dynamic thread balancing schemes on large SMP nodes.

4 Conclusions

By automatically varying the number of threads per MPI process based on timing results of the running program we offer an easy-to-use opportunity to improve the load balance of hybrid applications on clusters of shared-memory machines with fat nodes. The PMPI profiling interface of MPI allows us to easily capture relevant timing information, and as a result, a potential user only has to add a subroutine call in a few places of the code to trigger the dynamic thread balancing (DTB) mechanism.

Experimental results with the NAS BT parallel multizone benchmark and the FLOWer CFD solver show that use of the DTB library results in substantial improvements, already when just a few processors are added. In the case of the BT-MZ benchmark, the DTB library resulted in performance improvements comparable to that achieved by the BT-specific balancing mechanism provided by the benchmark developers, when the BT-specific load balancing mechanism could be employed. However, the programming effort required to use the DTB library is substantially less and it can be employed in a much more flexible fashion as the number of additional processors is concerned.

In future work, we will investigate to what extent we can take the ccNUMA architecture of the Sun Fire 15K architecture into account to migrate data closer to computational threads, and whether we can reduce overhead by decreasing the number of times we activate the DTB mechanism. We will also work on improving the robustness of the DTB mechanism during the warm-up phase and on loaded systems.

References

1. G. Jost, H. Jin, H., D. an Mey, and F. Hattay. Comparing the OpenMP, MPI and Hybrid Programming Paradigms on an SMP Cluster, *NAS Technical Report NAS-03-019*, NASA Ames Research Center, Moffet Field, CA, November 2003.
2. S.W. Bova, C.P. Breshears, H. Gabb, B. Kuhn, B. Magro, R. Eigenmann, G. Gaertner, S. Salvini, and H. Scott. Parallel Programming with Message Passing and Directives. *Computing in Science and Engineering*, September 2001, pp. 22-37.
3. R. Rabenseifner. Hybrid Parallel Programming on Parallel Platforms. *EWOMP'03 - Fifth European Workshop on OpenMP*, http://www.rz.rwth-aachen.de/ewomp03/omptalks/Tuesday/Session7/T01p.pdf.
4. M.K. Hesse, B. Reinartz, and J. Ballmann. Inviscid Flow Computation for the Shuttle-Like Configuration PHOENIX. *Notes on Numerical Fluid Mechanics*, Vol. 87, Eds. Chr. Breitsamter, B. Laschka, H.-J. Heinemann, R. Hilbig, Springer 2003, pp. 172-179.
5. M.K. Hesse, B. Reinartz, and J. Ballmann. Numerical Investigation of the Shuttle-Like Configuration PHOENIX. *High Performance Computing in Science and Engineering 2002*, Ed. E. Krause, W. Jäger, Springer Verlag, ISBN 3-540-43860-2, pp. 379-390.
6. M.K. Hesse, B. Reinartz, and J. Ballmann. Numerical Investigation of a Reusable Space Transportation System. *In: Proceedings of the 3rd International Symposium on Atmospheric Reentry Vehicles and Systems*, Arcachon/France, 24-27 March 2003.
7. D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. The NAS Parallel Benchmarks, *NAS Technical Report RNR-94-007*, NASA Ames Research Center, Moffet Field, CA, March 1994.

8. D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow. The NAS Parallel Benchmarks 2.0. *NAS Technical Report NAS-95-020*, NASA Ames Research Center, Moffet Field, CA, December 1995.
9. R.F. Van der Wijngaart, and H. Jin. NAS Parallel Benchmarks, Multi-Zone Versions. *NAS Technical Report NAS-03-010*, NASA Ames Research Center, Moffet Field, CA, July 2003.

Distributed Computing: Tools, Paradigms and Infrastructures. An Introduction

Organizers: Beniamino Di Martino[1], Rocco Aversa[1], and Laurence Tianruo Yang[2]

[1] Second University of Naples, Italy
[2] Francis Xavier University, Canada

An Overview

The technology and performance improvements of computational architectures and communication infrastructure, together with the increase of the demand for a greater and more easily usable computing power, generate new sophisticated requirements for distributed application development. The methodologies, infrastructures and tools traditionally used to develop distributed applications disclose a variety of limitations when applied to these new and evolving scenarios. The goal of this minisymposium is to investigate these issues, presenting some innovative approaches in managing the complexity in the design, development and performance analysis of distributed systems. Papers included in this minisymposium primary face the challenges of using the computing Grids as a commonly accepted general purpose computing platform, but the focus is also on paradigms and tools to support the programmers in developing efficient parallel solutions using MPI and OpenMP libraries. Furthermore, particular emphasis is given to some significant aspects of industrial and commercial distributed applications such as, the mobility, the dependability and the security.

Grid computing has become an area of extensive research. Major efforts are necessary to develop suitable parallel programming environments for computational grids.

Easy and convenient access of Grid systems is a foremost need for Grid end-users as well as for Grid application developers. Grid portals are the most promising environments to fulfill these requirements and Dózsa et al. decided to create a Grid portal for a Parallel Grid Runtime and Application Development Environment (P-GRADE) system. The portal allows users to manage the whole life-cycle of building and executing complex applications in the Grid: editing workflows, submitting jobs relying on Grid credentials and analyzing the monitored trace-data by means of visualization.

Mattson and Kessler describe research in progress on a new parallel programming environment for computational grids, called GridNestStep. It adopts a well established parallel programming model, namely bulk-synchronous parallel (BSP) computing,
which, by enforcing a clear organization of the program into parallel supersteps, provides an easily analyzable structure of GridNestStep programs.

In recent years, great effort has been made in developing methodologies and tools that could help programmers to develop applications independently of the underlying architecture, as in GRID environments; unfurtanatly, very few results have been obtained to support prediction and evaluation of prototypal applications. Mancini et al. propose a simulation-based methodology, based on HeSSE, a simulator of distributed applications executed in heterogeneous systems, and MetaPL, a prototype-based language based on

XML, able to support many different programming paradigms. This allows to predict GRID application and system performance, even when the execution environment is not available and the application is not completely developed.

Some papers included in this minisymposium present paradigms and tools to support the programmers in developing efficient parallel solutions.

Aliaga et al. employ two classical numerical computations to explore the challenges and difficulties involved in the parallelization of GNU Scientific Library (GSL), a collection of routines for numerical scientific computations. The present performance results for the parallelization of sparse matrix-vector product and for the Fast Fourier transform (FFT) with shared memory architectures, multicomputers and clusters of networked personal computers.

Gonzales et al. present a new MPI asynchronous peer-processor implementation of the MaLLBa:DnC skeleton where all processors are peered and behave the same way (except during the initialization phase) and where decisions are taken based only on local information. The MaLLBa:DnC skeleton solves problems that fit in the Divide and Conquer paradigm; its main objective is to simplify the implementation of algorithms based on some commonly used techniques such as Branch and Bound, Dynamic Programming or Divide and Conquer.

One way to improve OpenMP performance is to minimize the data sharing among threads, either by rearranging code or by privatizing data structures where possible. Prior to an effort of this kind, it is crucial to know how the arrays are being accessed by the executing threads, and detect which regions of arrays are shared between multiple threads. The idea of the paper presented by Hernandes et al. is to provide a visual environment to display the array access patterns detected at runtime. This information can help the programmer restructure their code to maximize data locality or to achieve OpenMP SPMD style code.

The minisymposium also includes some works on emerging aspects of industrial and commercial distributed applications.

The emerging technology to primary face integration and cooperation of services, is based on web services solutions, it is based on open standards and common data formats which allow a deep cooperation among entities and applications and guarantee strong resource sharing. In such context security plays a primary role to authenticate all subjects involved in any transaction and to guarantee correct authorizations to access data and functionalities offered by distributed services.

Casola et al. illustrate a policy-based approach to manage security and personalization issues in a distributed infrastructure based on web services, by presenting a centralized and distributed architecture to enforce security mechanisms and meet different security requirements.

Iacono et al. illustrate performability and dependability issues in distributed platform proposing a mobile agents monitoring system. They propose a multi-formalism approach based on model-checking techniques, queuing networks and timed Petri nets to model a real-time mobile agents-based monitoring system.

Di Flora et al. also have worked on mobile agent system and they propose a technique to estimate device location by evaluating the Received Signal Strength Indicator (RSSI). They present the effectiveness of the approach by providing preliminary experimental results obtained from our prototype system.

Parallelization of GSL: Performance of Case Studies*

José Aliaga[1], Francisco Almeida[2], José M. Badía[1], Sergio Barrachina[1], Vicente Blanco[2], María Castillo[1], U. Dorta[2], Rafael Mayo[1], Enrique S. Quintana[1], Gregorio Quintana[1], Casiano Rodríguez[2], and Francisco de Sande[2]

[1] Depto. de Ingeniería y Ciencia de Computadores, Univ. Jaume I, 12.071–Castellón, Spain
{aliaga,badia,barrachi,castillo, mayo,quintana,gquintan}@icc.uji.es

[2] Depto. de Estadística, Investigación Operativa y Computación, Univ. de La Laguna 38.271–La Laguna, Spain
{falmeida,vblanco,casiano,fsande}@ull.es

Abstract. In this paper we explore the parallelization of the scientific library from GNU both on shared-memory and distributed-memory architectures. A pair of classical operations, arising in sparse linear algebra and discrete mathematics, allow us to identify the major challenges involved in this task, and to analyze the performance, benefits, and drawbacks of two different possible parallelization approaches.

1 Introduction

The GNU Scientific Library (GSL) [4] is a collection of hundreds of routines for numerical scientific computations written in ANSI C, which includes codes for complex arithmetic, matrices and vectors, linear algebra, integration, statistics, and optimization, among others. The reason GSL was never parallelized seems to be the lack of a globally accepted standard for developing parallel applications. We believe that with the introduction of OpenMP and MPI the situation has changed substantially. OpenMP has become a standard *de facto* for exploiting parallelism using *threads*, while MPI is nowadays accepted as the standard interface for developing parallel applications following the message-passing programming model.

In this paper we investigate the parallelization of a specific part of GSL using OpenMP and MPI and their respective performances on shared-memory multiprocessors (SMPs) and distributed-memory multiprocessors (or multicomputers). In Section 2 we give a general overview of our parallel integrated version of GSL. Two classical numerical computations, arising in sparse linear algebra and discrete mathematics, are employed to explore the challenges involved in this task. In Section 3 we elaborate our first case study, describing a parallelization of the sparse matrix-vector product addressed to SMPs. In Section 4 the analysis is repeated for a parallel routine that computes the Fast Fourier Transform (FFT) targeted in this case for multicomputers. Finally, concluding remarks follow in Section 5.

* This work has been partially supported by the EC (FEDER) and the Spanish MCyT (Plan Nacional de I+D+I, TIC2002-04498-C05-05 and TIC2002-04400-C03).

2 Parallel GSL

Our general goal is to develop parallel versions of GSL using MPI and OpenMP which are portable to several parallel architectures, including distributed and shared-memory multiprocessors, hybrid systems, and clusters of heterogeneous nodes. Besides, we want to reach two different classes of users: a programmer with no experience in parallel programming, who will be denoted as user A, and a second programmer, or user B, that regularly utilizes MPI or OpenMP. As additional objectives, the routines included in our library should execute efficiently on the target parallel architecture and, equally important, the library should appear to user A as a collection of traditional serial routines. We believe our approach to be different to some other existing parallel scientific libraries (see, e.g., *http://www.netlib.org*) in that our library targets multiple classes of architectures while maintaining the sequential interface of the routines.

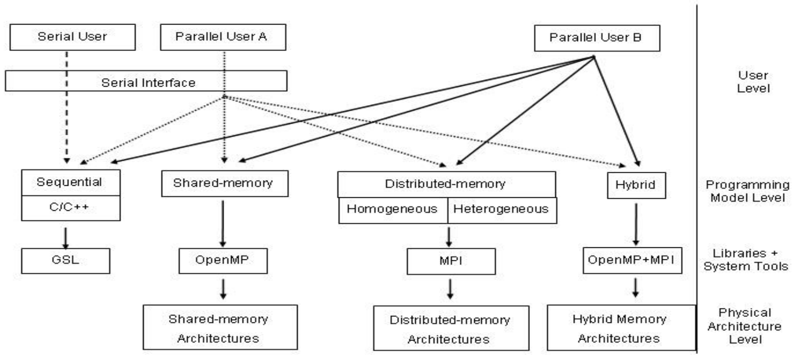


Fig. 1. Software architecture of the parallel version of GSL

Our library has been designed as a multilevel software architecture; see Fig. 1. The *User Level* (the top level) provides a sequential interface that hides the parallelism to user A and supplies the services through C/C++ functions according to the prototypes specified by the sequential GSL interface.

The *Programming Model Level* provides a different instantiation of the GSL library for each one of the computational models: sequential, distributed-memory, shared-memory, and hybrid. The semantics of the functions in the Programming Model Level are those of the parallel case so that user B can invoke them directly from her own parallel programs. The Programming Model Level implements the services for the upper level using standard libraries and parallelizing tools such as (the sequential) GSL, MPI, and OpenMP.

In the distributed-memory (or message-passing) programming model we view the parallel application as being executed by p *peer* processes, P_0, P_1,. . . ,P_{p-1}, where the same parallel code is executed by all processes on different data.

In the *Physical Architecture Level* the design includes shared-memory platforms, distributed-memory architectures, and hybrid and heterogeneous systems.

For further details of the functionality and interfaces, see [1].

3 Case Study I: Sparse Matrix-Vector Product

3.1 Operation

Sparse matrices arise in a vast amount of areas. Surprisingly enough, GSL does not include routines for sparse linear algebra, most likely due to the lack of an standardized interface definition, which was only developed very recently [3].

Here we explore the parallelization of the *sparse matrix-vector product*, denoted as USMV in the Level-2 sparse BLAS [3]. In this operation an $m \times n$ sparse matrix A, with nz nonzero elements, is multiplied by a vector x with n elements. The result, scaled by a value α, is used to update a vector y of order m as $y \leftarrow y + \alpha \cdot A \cdot x$.

The implementation, parallelization, and performance of the USMV operation strongly depends on the storage scheme employed for the sparse matrix. In the coordinate format, two integer arrays of length nz hold the row and column indices of the nonzero elements of the matrix, while a third array, of the same dimension, holds the values. In the columnwise variant of this format, the values are listed by columns. The rowwise Harwell-Boeing scheme also employs a pair of arrays of length nz for the column indices and the values. A third array, of length $m + 1$, determines then the starting/ending indices for each one of the rows of the matrix. Figure 2 illustrates the use of these two storage schemes by means of a simple example.

A 3x4 sparse matrix

	0	1	2	3
0	1.0	2.0		
1				4.0
2		3.0		

Columnwise coordinate

	0	1	2	3
rows	0	0	2	1
cols.	0	1	1	3
values	1.0	2.0	3.0	4.0

Rowwise Harwell–Boeing

	0	1	2	3
rows	0	2	3	4
cols.	0	1	3	1
values	1.0	2.0	4.0	3.0

Fig. 2. Storage schemes for a 3×4 sparse matrix with $nz = 4$ nonzero elements

The USMV operation is usually implemented as a series of *saxpy* (scalar α times x plus y) operations or *dot products* depending respectively on whether the matrix is stored columnwise or rowwise. In particular, in the columnwise implementation of the matrix-vector product, for each column of A, denoted as $A(:, j)$, a saxpy operation is performed to update vector y as $y \leftarrow y + A(:, j) \cdot (\alpha * x[j])$. In the rowwise case, the matrix-vector product is computed as a series of dot products, involving a row of matrix A, denoted as $A(i, :)$, and x. The result is used to update a single element of y as $y[i] \leftarrow y[i] + \alpha \cdot (A(i, :) \cdot x)$. Codes for the columnwise/rowwise implementation of the sparse matrix-vector product are given in Fig. 3. Minor modifications are introduced in those codes to facilitate the exposition of the parallel implementations using OpenMP.

3.2 Parallel Implementation Using OpenMP

Parallelizing a sequential code using OpenMP is, in most cases, straight-forward. One only needs to evaluate possible data dependencies and then place the appropriate directives in loops/regions of the code that indicate the OpenMP environment how to extract

```
indval=0;
for(j=0;j<n;j++){
  nzj=nz_in_column(j);
  temp=alpha*x[j];
  for(i=0;i<nzj;i++){
    k=rows[indval+i];
    y[k]+=values[indval+i]
    *temp;
  }
  indval+=nzj;
}
```

```
for(i=0;i<m;i++){
  indval=rows[i];
  temp=0.0;
  for(j=rows[i];j<rows[i+1];
    j++){
    k=cols[indval];
    temp+=values[indval]
    *x[k];
    indval++;
  }
  y[i]=y[i]+alpha*temp;
}
```

Fig. 3. Implementation of the USMV operation for the columnwise coordinate format (left) and the rowwise Harwell-Boeing format (right). Routine nz_in_column returns the number of nonzero elements in a column

parallelism from the corresponding parts using threads. As a general rule, in codes composed of nested loops it is better to parallelize the outermost loop as that distributes a larger part computational load among the threads.

Consider now the columnwise coordinate implementation of USMV (left of Fig. 3). Parallelizing the outer loop in this code implies that two or more threads could be concurrently updating the same element of vector y, which is a well-known case for a race condition. A solution to this problem is to guarantee exclusive access to the elements of y; however, synchronization mechanisms for mutual exclusion usually result in large overheads and performance loss. Therefore, we decide to only parallelize the inner loop in this case so that the outer loop is executed by a single *master* thread, while multiple threads will cooperate during the computation of each saxpy operation performed in the inner loop. In order to do so, we need to introduce the OpenMP directive:

```
#pragma omp parallel for private (i, k)
```

just above the inner loop. Given that each iteration of the inner loop requires the same amount of computations, a static scheduling of the threads achieves a good distribution of the computational load for this implementation. The parallelization of the inner loop, though simple, is not completely satisfactory. In general, the number of nonzero entries per column is small, while the number of columns of the matrix is much larger. Since the threads need to be synchronized every time execution of the inner loop is terminated (that is, once per column), a large overhead is again likely to arise in this approach.

The parallelization of the rowwise implementation (right of Fig. 3) using multiple threads is much easier. As each iteration of the outer loop is independent, we can parallelize this loop by introducing the directive:

```
#pragma omp parallel for private (indval, temp, j, k)
```

just before the outer loop. A certain amount of dot products are thus computed by each thread and the threads only need to be synchronized once, on termination of the outer loop. A dynamic scheduling of the threads provides a better computational load balancing in this case as usually the nonzero entries of the matrix are distributed unevenly among the rows.

Table 1. Time (in sec.) of the parallel implementations for the USMV operation

Density	Columnwise coordinate			Rowwise Harwell-Boeing		
	1 Thread	2 Threads	4 Threads	1 Thread	2 Threads	4 Threads
0.01	0.031	1.961	0.485	0.029	0.021	0.012
0.1	0.292	1.676	0.608	0.207	0.162	0.097
1.0	2.855	2.389	1.390	1.976	1.560	0.917

3.3 Experimental Results

All the experiments in this subsection were obtained on a 4-way SMP UMA platform consisting of 4 Intel Pentium Xeon@700MHz processors with 2.5 Gbytes of RAM and a 1 Mbyte L3 cache. We employed the Omni 1.6 portable implementation of OpenMP for SMPs (`http://phase.hpcc.jp/Omni/`).

We report the performance of the parallel implementations of the USMV operation for a sparse matrix with $m = n = 50,000$ rows/columns and a rate of nonzero elements (density) ranging from 0.01$\%$ to 1$\%$. The sparse matrices are generated so that the elements are randomly distributed among the rows/columns of the matrix resulting in a similar number of nonzero elements per row/column. The number of (floating-point arithmetic) operations of the sparse matrix-vector product is proportional to the square of the number of nonzero entries of the matrix, and therefore the computational cost of this problem can be considered as moderate. In other words, one should not expect large speed-ups unless the matrix becomes “less sparse”. However, sparse matrices with a density rate larger than 1$\%$ are rare.

Table 1 shows the execution times of the sequential code (columns labeled as 1 thread) and the parallel columnwise/rowwise codes using 2 and 4 threads. Comparing only the sequential codes, the columnwise implementation obtains larger execution times. This is usual as current architectures, where the memory is structured hierarchically in multiple levels, benefit more from an operation such as the dot product than saxpys. One could then argue against the use of the columnwise code. However, notice that once the storage scheme of the matrix is fixed as rowwise, the computation of $y \leftarrow y + \alpha \cdot A^T \cdot x$, an operation probably as frequent as the non-transposed matrix-vector product, requires accessing the elements of the matrix as in the columnwise code.

Now, let us consider the parallel performances of both variants. The columnwise implementation offers poor parallel results until the density of nonzero elements reaches 1$\%$. This behaviour was already predicted in the previous subsection: As the threads need to be synchronized once per column, when the number of nonzero entries per column is small compared with the number of columns of the matrix, the overhead imposed by the synchronization degrades the performance of the algorithm. Only when the density rate is 1$\%$ the parallel implementation outperforms the serial code, with speed-ups of 1.19 and 2.05 for 2 and 4 threads, respectively.

For the rowwise implementation, the experimental results show maintained speed-ups around 1.3 and 2.4 using 2 and 4 threads, respectively, for all density rates. Further experimentation is needed here to evaluate whether a different implementation of OpenMP, or even a different platform, would offer better performances.

4 Case Study II: FFT

4.1 Operation

The Fast Fourier Transform (FFT) plays an important role in many scientific and engineering applications arising, e.g., in computational physics, digital signal processing, image processing, and weather simulation and forecast.

We analyze hereafter the parallelization of the FFT routines in GSL. The case when the number of processors is a power of two has been extensively analyzed, and we employ here the version in [5]. The FFT algorithms can be generalized for the case when neither the number of processors nor the problem size, n, are a power of two. However, in general, this introduces a non-negligible communication overhead and results in a considerable load imbalance. We follow the parallel mixed-radix FFT algorithm presented in [2], which is a variant of Temperton's FFT [6]. The algorithm presents a low communication overhead, has good load balance properties, is independent of the number of processors, and poses no restrictions on the number of processors nor the size of the input vector.

We start by introducing some preliminary notation needed for the description of the FFT algorithm [2]:

- Rows and columns of matrices are indexed starting from 0.
- Element (j, k) of matrix A is stated to as $\mid A \mid (j, k)$.
- The Kronecker product of matrices A and B is denoted as $A \otimes B = (a_{ij}B)$.
- The permutation matrix P_q^p of order pq is defined as

$$\mid P_q^p \mid (j,k) = \begin{cases} 1 & \text{if } j = rp + s \text{ and } k = sq + r, \\ 0 & \text{otherwise.} \end{cases}$$

- The diagonal matrix D_q^p of order pq is defined as

$$\mid D_q^p \mid (j,k) = \begin{cases} w^{sm} & \text{if } j = k = sq + m, \\ 0 & \text{otherwise,} \end{cases}$$

  where $w = \exp(2\pi\imath/pq)$, $\imath = \sqrt{-1}$.
- I_m states for the square identity matrix of order m.

By definition, the Discrete Fourier Transform (DFT) of $z = (z_0, ..., z_{n-1})^T \in \boldsymbol{C}^n$ is given by

$$x_j = \sum_{k=0}^{n-1} z_k \exp(2\pi jk\imath/n), \quad 0 \le j \le n, \quad x \in \boldsymbol{C}^n. \tag{4.1}$$

Alternatively,

$$x = W_n z, \quad W_n(j,k) = w^{jk}, \quad w = \exp(2\pi\iota/n). \tag{4.2}$$

Here, W_n is known as the DFT matrix of order n. The idea behind the FFT algorithm is to factorize W_n so that the multiplication by W_n is broken up into $O(n \log n)$ multiplications involving smaller matrices (each one of a small constant size). Temperton presented in [6] several variants of the FFT algorithm by rearranging the terms as different factorized multiplications of the form $W_n z$. In particular, for $n = f_1 f_2 \cdots f_{k-1} f_k$,

$$W_n = T_k\, T_{k-1} \cdots T_2\, T_1\, P_1\, P_2 \cdots P_{k-1}\, P_k, \tag{4.3}$$

with

$$T_i = (I_{m_i} \otimes W_{f_i} \otimes I_{l_i})\,(I_{m_i} \otimes D^{f_i}_{l_i}), \quad P_i = (I_{m_i} \otimes P^{l_i}_{f_i}),$$

$l_i = 1, l_{i+1} = f_i l_i, m_i = n/l_{i+1}$, and $1 \le i \le k$. Notice that all permutation matrices in (4.3) are shifted to the right. From this formulation we arrive at the following sequential algorithm for the FFT, with complexity $O(n \sum f_i)$:

```
Factorize(n, k);  // Compute f[0], ..., f[k-1]
for (i = k-1; i >= 0; i--)
  z = P[i] * z;
for (i = 0; i < k; i++)
   z = T[i] * z;  // m[i]*l[i] sub-DFTs of size n
                  // have to be solved
```

While the first loop computes a permutation of the input array, the parallelization effort is focused in the second loop.

4.2 Parallel Implementation Using MPI

The parallelization of the above algorithm has been also presented in [2] and we describe it with the code below. The algorithm is based on a particular layout of the data to reduce the communication overhead with a proper load balance. At the beginning of each iteration, processor p_j receives just the data needed for the computation during this iteration, according to a Rolled_Break_Cyclic distribution (to be described later; see fig. 4). Only f_i communication steps among processors $p_j - 1$ and $p_j + 1$ are involved. After the computation of the local subproblems, the Reverse_Distributed_Rolled_Break_Cyclic routine reallocates the data according to a pure cyclic layout.

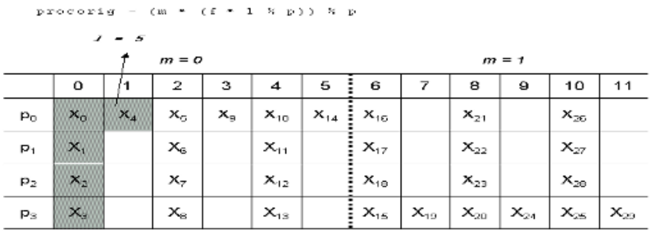
```
Factorize(n, k);     // Compute f[0], ..., f[k-1]
Distribute_Cyclic(); // The permutation of the
                     // input array is distributed
                     // following a cyclic distribution
l = 1;               // l = l[i]; m = m[i]
for (i = 0; i < k; i++) {
  m = n / (f[i] * l);

  Distribute_Rolled_Break_Cyclic(l, m, f);
                      // m[i] * l[i]/p sub-DFTs of size
                      // f[i] have been distributed on
                      // each processor

  for (j = 0; j < m * l /p; j++) {
    v[j] = D * v[j];  // v[j] is the local vector
                      // corresponding to sub-DFT j
                      // D is the diagonal matrix of order
                      // f[i]l. Submatrices of D of order
                      // f[i]f[i] are involved in each
                      // product

    v[j] = Wf * v[j]; // Wf is the DFT matrix of order f
  }
  Reverse_Distributed_Rolled_Break_Cyclic(l, m, f);
  l = l * f[i];
}
```

A total number of lm sub-DFTs of size f must be solved in iteration i. The perfect load balance of the work is found when the lm sub-DFTs can be evenly distributed among the actual number of processors. The `Rolled_Break_Cyclic` distribution groups them into m blocks of size l among the processors. Each block has a representative processor, `procorig`, from which the allocation of data starts in the block. A cyclic assignment is then performed. After l data elements have been allocated, the cyclic distribution breaks and continues again starting from `procorig`. The balanced load of work is attained since `procorig` receives $\lfloor l/p \rfloor + 1$ sub-DFTs and the remaining processors receive $\lfloor l/p \rfloor$ sub-DFTs. Figure 4 depicts a `Rolled_Break_Cyclic` distribution of $n = 30$ elements on $p = 4$ processors. The number of blocks is $m = 2$, and the number of sub-DFTs, per block, $l = 5$, is divided among the processors. In block $m = 0$ processor p_0 is `procorg` and is assigned 2 sub-DFTs, with the first one composed by the elements $\{x_0, x_5, x_{10}\}$ and the second one consisting of $\{x_4, x_5, x_{14}\}$. The remaining processors receive 1 sub-DFT in this block. In block $m = 1$ `procorig` is processor p_3.


procorig = (m * (f * l % p)) % p
l = 5
m = 0
m = 1

	0	1	2	3	4	5	6	7	8	9	10	11
p_0	x_0	x_4	x_5	x_9	x_{10}	x_{14}	x_{16}		x_{21}		x_{26}	
p_1	x_1		x_6		x_{11}		x_{17}		x_{22}		x_{27}	
p_2	x_2		x_7		x_{12}		x_{18}		x_{23}		x_{28}	
p_3	x_3		x_8		x_{13}		x_{15}	x_{19}	x_{20}	x_{24}	x_{25}	x_{29}

Fig. 4. A Rolled Break Distribution of $n = 30$ elements on $p = 4$ processors. $l = 5$, $f = 3$ and $m = 2$

4.3 Experimental Results

The experiments presented in this subsection were performed on a PC Cluster consisting of a 32 Intel Pentium Xeon running at 2.4 GHz, with 1 GByte of RAM memory each, and connected through a Myrinet switch. We have used the MPI implementation GMMPI 1.6.3. Vectors with complex entries randomly distributed were generated as input signals for the algorithm. Two instance problem sizes are considered in the experiments, $n =$

$510,510$ and $n = 1,048,576$. The first one is not a power of two. An interesting feature is shown in Fig. 5, where the variation of the communication cost for the FFT algorithm is reported. For both problem sizes, the communication volume remains almost constant when the number of processors is increased, that is, the total time invested in communications is not incremented as more processors are added. Table 2 shows the execution times for the FFT algorithm. Although the first instance size is not a power of two, the curves show a similar behaviour for both problem dimensions: execution times decrease as more processors are added to the experiment. The load balance achieved by the `Rolled_Break_Cyclic` distribution is strongly dependent on the factors f_i selected so that better performances can be expected if the factorizations are adapted to the problem size and the number of processors but the computational effort may be prohibitive.

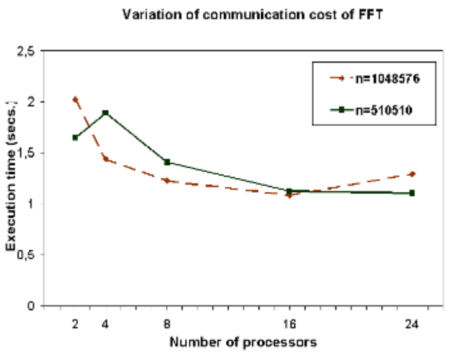


Fig. 5. Variation of the communication cost of the FFT algorithm

Table 2. Time (in sec.) of the parallel implementations for the FFT operation

	Number of processors					
Problem Size	1	2	4	8	16	32
510,510	9.80	8.45	5.27	3.40	2.66	2.16
1,048,576	16.30	14.21	8.95	5.99	4.56	3.67

5 Concluding Remarks

We have described our efforts towards an efficient and portable parallelization of the GSL library using OpenMP and MPI. Experimental results on an SMP and a multicomputer report the performance of several parallel codes for the computation of two classical, simple operations arising in linear algebra and discrete mathematics: the sparse matrix-vector product and the FFT.

In particular, we have experienced that parallelizing the codes for the computation of the sparse matrix-vector product is simple, but the performance depends not only on the algorithm but also on the matrix storage scheme, sparsity pattern, and density rate. Although the scalability of shared-memory architectures is intrinsically limited, experience taught us that in a parallelization of the sparse matrix-vector product on a multicomputer, the communication time plays an important role, becoming a major bottleneck.

We have also observed that the parallelization of the FFT general algorithm is quite elaborate. Although many parallel algorithms have been implemented on dozens of parallel platforms, most of them use internal code optimizations to run efficiently both sequentially and in parallel. Our own FFT codes should then make use of similar optimization techniques to be competitive at the cost of loosing portability. Better performances are expected in shared-memory architectures, where the communication overhead can be avoided, but the load imbalance inherent to the distribution of the sub-DFTs in the parallel algorithm seems difficult to avoid.

References

1. J. Aliaga, F. Almeida, J. M. Badía, V. Blanco, M. Castillo, U. Dorta, R. Mayo, E.S. Quintana, G. Quintana, C. Rodríguez, and F. de Sande. Parallelization of gsl: Architecture, interfaces, and programming models. In D. Kranzlmüller, P. Kacsuk, and J. Dongarra, editors, *EuroPVM/MPI 2004*, number 3241 in Lecture Notes in Computer Science, pages 199–206. Springer-Verlag, Berlin, Heidelberg, New York, 2004.
2. G. Banga and G. M. Shroff. Communication efficient parallel mixed-radix FFTs. Technical Report 94-1, Indian Institute of Technology, February 1994.
3. I.S. Duff, M.A. Heroux, and R. Pozo. An overview of the sparse basic linear algebra subprograms. *ACM Trans. Math. Software*, 28(2):239–267, 2002.
4. M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *GNU scientific library reference manual*, July 2002. Ed. 1.2, for GSL Version 1.2.
5. M. J. Quinn. *Parallel Computing. Theory and Practice*. McGraw-Hill, 1994.
6. C. Temperton. Self-sorting mixed-radix fast fourier transforms. *Journal of Computational Physics*, 52:1–23, 1983.

Design of Policy-Based Security Mechanisms in a Distributed Web Services Architecture*

Valentina Casola[1], Antonino Mazzeo[2],
Nicola Mazzocca[2], and Salvatore Venticinque[1]

[1] Seconda Universita' di Napoli
Dipartimento di Ingegneria dell'Informazione
Aversa (CE), Italy
{valentina.casola,salvatore.venticinque}@unina2.it
[2] Universita' degli Studi di Napoli, Federico II
Dipartimento di Informatica e Sistemistica
Naples, Italy
{mazzeo,n.mazzocca}@unina.it

Abstract. In the recent years, modern complex infrastructures are built on integration and cooperation of legacy and/or new systems; the emerging technology, to primary face the involved interoperability problems, is based on web service solutions. It is based on open standards and common data formats which allow a deep cooperation among Entities and applications and guarantee strong resource sharing. In such context security plays a primary role to control access to data and functionalities offered by distributed services. In this paper we illustrate a policy-based approach to manage security and personalization, in particular we have designed a hybrid infrastructure based on web services in which policy enforcer mechanisms are managed both in a centralized way by the registry server and in a distributed way, i.e. each service implements security mechanisms to authenticate and authorize users. A case study is illustrated showing a distributed architecture for health-care applications.

1 Introduction

In the recent years we are assisting to a wide number of experiences in the integration and cooperation of legacy and/or new systems; in general, complex transactions involve the integration of services managed by different Entities. Sometimes the integration is performed by a human worker who is able to access the different providers and to complete each transaction by a special interface. However, the integration should be handled by software applications that can automatically access the system and then exploit the required services. In the latter situation we distinguish two different approaches:

1. All Providers are forced to observe a set of fixed rules conceived by a centralized authority in order to grant interoperability and to provide common mechanisms for accessing services and resources.

* This work was partially supported by “Centro Regionale di Competenze" and “Progetto Telemedicina" Regione Campania.

2. There are no restraints about the implementation of services. Each provider has just in charge the definition of its own policies in such a way that Security Level Agreements and Quality of Services constraints are satisfied for the global transaction.

In order to realize a cooperative system we need a physical infrastructure that makes the facilities accessible by software applications from remote requestors. The main requirements for such kind of systems are interoperability, heterogeneity support, capability to adopt and integrate legacy systems. The emerging infrastructure to primary face interoperability problems and to efficiently meet the listed requirements, is based on Web Service solutions [5]; they are in fact based on open standards and common data formats which allow a deep cooperation among Entities and applications and guarantee strong resource sharing. In such context, security plays a primary role to authenticate all subjects involved in any transaction and to control the access to data and functionalities offered by distributed services. Usually, each service manages its own security mechanism. When we integrate different services, how a security manager could manage the global security? To address this problems, we have first introduced the adoption of policy-based approaches which are able to split security and application specific problems, then, we have investigated two security models to manage policies to finally propose a hybrid security mechanism. The reminder of this paper is structured as follows: details about Security mechanisms in Web Service architectures are reported in Section 2 while in Section 3 we will focus our attention on design principles which should be taken in count to enforce and manage proper security policies and mechanisms in a distributed infrastructure; a case study is illustrated in Section 4 showing an implementation of a prototypal architecture for the application of our approach for healthcare services.

2 Providing Security in Web Services Architectures

Web services are an emerging distributed technology which focuses on Internet-based standards technology such as XML, HTTP, SOAP, WSDL and so on [6]; they have been conceived to support interoperability, i.e. independence of the transport protocols, programming language, programming models and system software. The main actors in a Web Service architecture are requestors and providers. The first ones access service providers acting as clients in order to provide new value added services to the final users. The added-value can deal with the best suited service to users' profile, with the cheaper one or with the integration of more services in a wide transaction. A registry server, implemented by the Universal Description Discovery and Integration (UDDI) technology [12], allows the requestor to look for available services, which are delivered by the different providers. The integration is performed at application level by the requestor that is able to access external services according to a specific workflow. Web Service is a widely accepted standard for interoperability among web oriented multi-tier applications and it seems to be plain how the system design has to be approached and what technologies have to be involved. However it is not clear how a common security infrastructure should be built in order to grant some basic requirements [3,9] and available solutions are not mature, yet. Security problems need to be addressed very carefully as security “threats" are very subtle, furthermore they act at all levels of the Infrastructure.

For these reasons, there is the need of designing more than just one mechanism, and there is the need to properly design, implement and manage them at all levels [2]. In this paper we have focused our attention on security at application level, in particular on:

– Identification and Authentication,
– Access Control: Authorization.

In distributed systems, each transaction needs to be traced from the origin to its completion, across different domains or tiers, without losing its security level; at this aim we intend to propose not a proprietary technique but a policy-based approach, as it allows an efficient and flexible management of security. It is based on two distinct elements:

– a *policy*, that is a set of security rules which are able to express security statements on *who* (a single user, a group, a special role,..) can access a *resource* and which *actions* are allowed.
– a *policy enforcer engine* which is enforced by a resource manager to protect the resource (functionalities and data of a generic service).

The reminder of this paper is focused on design principles to implement policy-based security mechanisms in a complex web-services architecture.

3 Design of Security Mechanisms

In general, security problems involve both technical and management issues. In fact, when we talk about security technical issues, we rely not only on mechanisms to protect the infrastructure and all services/resources from malicious or not-authorized users, but also on organizational aspects which are subtler and that need to be managed by proper security policies. In this paper we will focus our attention on Web Service and introduce two architectural choices:

1. centralized security mechanisms, enforced on the Access Point,
2. distributed security mechanisms, enforced on each single service provider.

In the next subsections we will illustrate these two design solutions and our novel hybrid approach.

3.1 Centralized Mechanisms

In a centralized approach, when an external user (or other external services) asks for a service, the Access Point implements Resource Manager functionalities enforcing the policy engine. In this case the Access Point is responsible for implementing authentication mechanisms (weak or strong); for example, a weak authentication method (login/password based) verifies whether the user login is in the user-DB (identification) and the corresponding password matches the one in the passwd-DB (authentication). If these steps are successfully completed, the Policy Manager performs the authorization process.

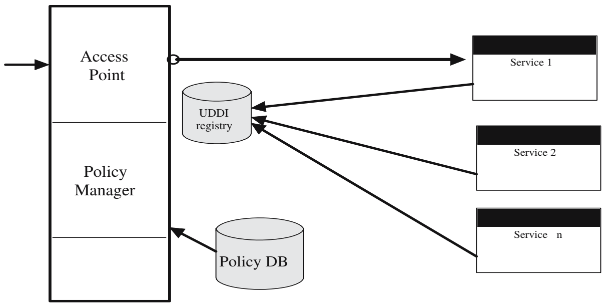


Fig. 1. Centralized security mechanisms

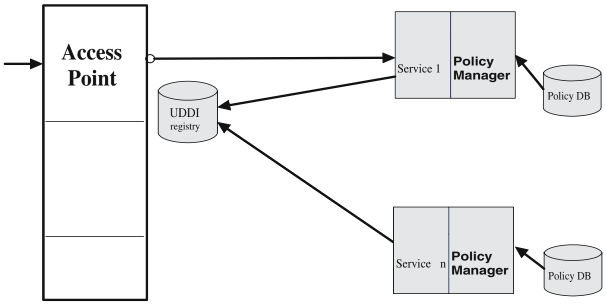


Fig. 2. Distributed security mechanisms

The request is forwarded to a Provider registered in the UDDI server only if the Policy manager has granted the claimed authorization (Figure 1).

The Service Provider itself needs to authenticate the Access Point and to process its authorizations before granting and serving the request. If the communication between the service provider and the Access Point uses a secure channel the request needs to include just the proper parameters and the service invocation request. If the communication uses a public channel then we need to provide other security mechanisms acting at different levels (for example SSL, Kerberos).

In conclusion, the centralized method is conceptually very simply to implement; the access evaluation function is enforced on the Access Point to manage the global security; the real drawbacks are the growing of database size (data and policies) and the centralized management of different security domains.

3.2 Distributed Mechanisms

In a distributed approach, the Access Point forwards the access request to the service provider which locally implements all security mechanisms, as shown in Figure 2.

Practically, the authentication and authorization evaluation model is the same of the centralized way but the policy manager is locally managed and enforced, so we would not spend many words on this, indeed, what we want to underline here is that, locally,

a security administrator could enforce a very fine-grain policy. This implies that it is possible to manage a very fine access control based on credentials. Furthermore it is possible to personalize each request according to users' profiles, even on the single user and not just on his group/role. The personalization is important for two primary reasons: to manage different client technologies and, furthermore, to manage a fine grain access control to resources [4]. Indeed, defining very fine grain access rules and policies makes possible to associate a very detailed access profile to users and so to apply very restrictive policies not just on functionalities but on data, too. For example suppose you need to implement the following rule:
“Only the medical doctor of the patient X affected by AIDS could read and modify his medical records".
This is a very simple example of rule which could not be enforced with the simple use of high-level credentials such as the role “medical doctor" who can, in general, modify a clinical record. The privacy issue involved in the specific pathology (AIDS) puts in relation both the subject who wants to “act on" the resource (modify) and some “attributes" of the resource itself (who is the subject of the records? and who is the doctor of the subject?).

In order to make possible the implementation of such rules, we want to propose a mechanism to make service provider able to delegate the access control of his services to other trusted providers when a fine grain authentication is not required. At the same time the service provider is able to locally perform the access control when a fine grain control is required.

3.3 Our Approach

The main goal of this paper is to propose a hybrid security mechanism to overcome the limitation of classical models (e.g. the pure centralized one and the pure distributed one) and to allow the exploitation of more flexible authentication and authorization mechanisms.

The classical distributed mechanism allows a fine-grain access control, but the overhead of authentication and authorization must carried on by the service provider. In the centralized mechanism the providers have to delegate the access control to others (in our design, to the Access Point) and this is not so easy to manage in a large distributed environment.

We propose a hybrid model that allows the provider to define and expose its authorization policy in order to delegate authentication and authorization to a trusted Third Party (the Access Point). Furthermore, when the service exploitation requires a finer access control the requestor will include clients' credential in the message that will be locally processed and authorized by the Provider which could apply more restrictive policies.

We have developed a security infrastructure that works as follows:

1. The service provider defines a role based security policy for a subset of its services which need a *coarse grain* authorization. The service provider defines a *fine grain* policy for all the other services;
2. The service provider trusts a third party that is able to download the policy and to access any service belonging to the first subset;

3. The trusted party authenticates the users and it performs the *coarse grain* authorization;
4. If these steps are successfully completed, the Policy Manager gathers the credentials (typically a role or a group) of the user and downloads the policy of the requested service as shown in Figure 3, in order to verify if the claimed request is allowed to him. Finally the request is submitted to the provider that holds the invoked service.
5. When the user invokes a service belonging to the first subset, the trusted party access the service; when the user invokes a service belonging to the second subset the trusted party access the service forwarding users credentials in the request itself (for example X.509 certificate) and access control is locally performed.

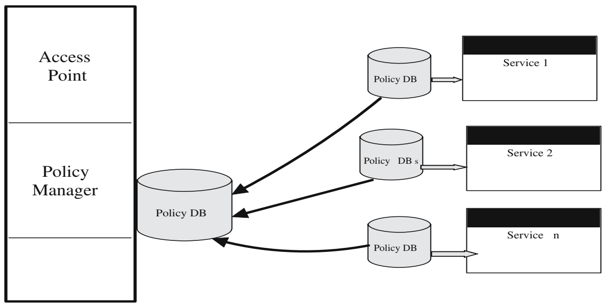


Fig. 3. Hybrid security manager

Service providers have to define policies, and let them be available for download from the trusted party. In this case the service provider must just authenticate this party and trust any access from it.

4 Case Study

Having introduced the general design of security mechanisms, we are now concerned with a closer look onto the implementation of a prototypal architecture with a hybrid mechanism applied on web services infrastructures for health-care applications. In health-care applications, there is the need to control access to private data (e.g. medical records) and specific functionalities. At this aim we have first defined different users typologies and roles and then we have written different access control rules to adopt the Role-based Access Control model [7], then we have implemented the applications and defined the operations that each role could acts on them. The chosen policy language is the XML Access Control Language (XACL) [8], by which security rules are expressed by the ternary vector (role, resource, operation). XACL is quite similar to the standard eXtensible Access Control Markup Language (XACML) [11], its expression power is limited, but a free policy enforcer engine is available and it is suitable for our prototype. We have defined 4 different user typologies:

role	resource	operation
primary doctor	registry records	“read"
primary doctor	analysis records	“read" and “write"
primary doctor	therapy records	“read" and “write"
assistant doctor	registry records	“read"
assistant doctor	analysis records	“read"
assistant doctor	therapy records	“read" and “write"
therapy doctor	registry records	“read"
therapy doctor	analysis records	“read"
therapy doctor	therapy records	“read" and “write"
administrators	registry records	“read" and “write"
administrators	analysis records	-
administrators	therapy records	-
patients	registry records	“read"
patients	analysis records	“read"
patients	therapy records	“read"

Fig. 4. Access Control Rules

medical personnel; administrators; patients; generic users;

In each typology we have located different roles, for example the roles involved in the medical personnel one are: *primary doctor, assistant doctor, therapy doctor and nurse*. As already said, they were adopted for the implementation of the Role based Access Control model (RBAC) [7]. Our critical application consists of the management of *medical records*; in particular, the implemented functionalities are the basic operations (“read" and “write"), while the resources (that is the sensible data) are the specific fields of the *medical record*. In our prototype we have assumed the resources are: *registry records, analysis records and therapy records*, which have different privacy issues and security requirements.

In Figure 4 we report an useful representation of some access control rules, in it each column represents an element of the vector (role, resource, operation) and each raw represents a rule.

It is important to underline that the management of user profiles is completely independent from the Web Services Security layer but it is exploited in each transaction. We have defined different profiles for authorization and for different device technologies; so, we have also adopted this information to early discover the best service available for a user according to both his authorization and technological profile.

4.1 The Provided Implementation

In our implementation the published services provide a method to download vendor's policy. The prototype is illustrated in Figure 5. Three providers offer some services

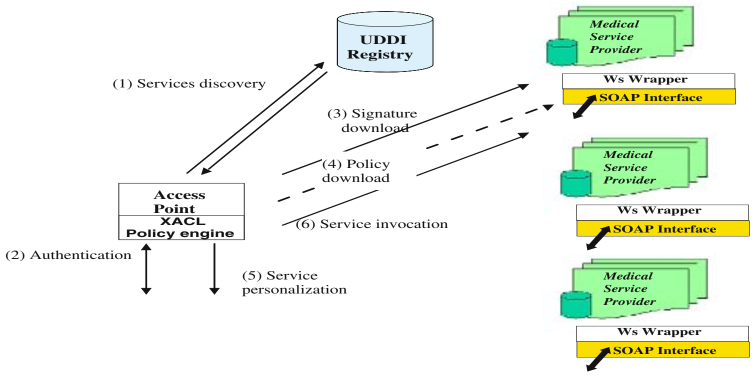


Fig. 5. Service exploitation in the implemented prototype

to their customers, who need to be subscribed to the deployed services. The subscriptions, the roles and the authorization rules for each service are defined by each provider autonomously. Each time the policies are upgraded they are signed, stored and made available to be downloaded by the policy enforcer engine. The providers deploy their services in a public or private UDDI register. The Access Point hosts a web portal that provides an integrated web access to distributed Health Care services. It provides three different authentication mechanisms (i.e. login/passwd, hashed challenge response and authentication by means of X.509 digital certificates [10,1]) and provides authorization by an XACL policy enforcer engine developed by the IBM [8].

The authorization engine retrieves user's credentials from the certificate (according to an Italian law we put the role information in the *subject* field of the X.509 certificate) and it gathers the signed policies from remote providers. According to the authorization results it is possible to personalize the access by presenting to the user just the allowed services and functionalities. In the following we describe the logical sequence for service exploitation.

After an user request, the Access Point is able to discover the allowable services searching for them in the register and to present them to the user (1);the research result is a list of available providers offering the requested service. The first time the user asks for a service, beyond a successful authentication (2), the Access Point is able to retrieve the signature of the policy (3) and to check if it is still consistent with the owned one. If the signatures do not match a new policy is downloaded from the provider (4). After the evaluation of the authorizations, the service is presented to the user with the specific functionalities which are allowed to him (5). This choice allows the provider to delegate to an external web portal the presentation and the personalization of its services without to relax the relationships with their customers, and preserving the full control of its own security policies. In fact the users do not need to establish any other relationship with other third parties (i.e. the Portal) in order to exploit the services. This aspect is important because, for example, a doctor could be employed in different hospitals assuming the same or different roles but with different authorization rules, since each provider defines its own policies. In the same way a patient could be a customer of different hospitals,

and be registered with their services, according to its own needs. About the selected case study, a primary doctor could look for his patients' records, and will be able to read or to modify the information. In the same way a patient will be able just to look for his data to read and not to modify them. In order to support interoperability between any provider and the Access Point, we deployed the services by the Web Services technology.
The *execution environment* is the J2SE 1.4.0 and the application platform is J2EE integrated with the JWSDP 1.0.01 (The Java Web Services Developer Pack). JWSDP 1.0.01 provides: the Java XML Pack to support XML-based applications and the SOAP protocol, a tool for Web Services deploying an application server (Tomcat 4.0), a registry server that implements the UDDI register named Xindice. We have installed the *application environment* on each provider to deploy the services and on the registry server in order to allow publication and discovery in the UDDI register. The Access Point integrates Apache2.0 and Tomcat4.0 to adopt the suite of implemented servlets and Java Server Pages (JSP), which exploit a set of high level APIs to access to UDDI register, and to invoke XACL mechanism. As we have enforced security mechanisms on the Access Point, we have assumed that the nodes of the Infrastructure (the web portal, its components, the DB serve and the service providers) are connected by a virtual private network (VPN) that provides security at network and transport layers and allows the authentication among the nodes. In a different application context, for example e-commerce and marketplace, the Access Point could offer the personalization as a value-added service by defining new policies and profiles for its customers; in this case the new security engine should process a combination of the local and the remote policies to guarantee security and personalization, too.

5 Conclusions and Future Works

In this paper we have described different approaches for designing security mechanisms in Web Service architectures and we have proposed a novel hybrid security mechanisms in order to efficiently manage security issues. We have presented a Health Care case study and developed a prototypal implementation that uses the hybrid model to provide security and personalization for web services in Telemedicine Projects. We have adopted a proprietary UDDI register, so in an early version of our system the security enforcer engine is integrated in the Access Point and not directly in the register. Actually we are implementing an extended model of the UDDI register which is integrated with the enforcer and is able to process profiled requests, too.

References

1. G.B. Barone, N.Margarita, N.Mazzocca, A.Mazzeo and L.Romano. Secure Access to Personalized Web Services. Proceedings of Pacific Rim International Symposium on Dependable Computing. IEEE Comput. Soc, pp. 266-269. Seoul, Korea, 2001.
2. M.S. Baum, W. Ford. Secure Electronic Commerce. *Ed. Prentice Hall, 1997.*
3. K. Beznosov, B. Hartman, D.J. Flinn, and S Kawamoto. Mastering Web Services Security. *Wiley.*

4. S. Boll. Modular Content Personalization Service Architecture for E-Commerce Applications. Proceedings of the 4th IEEE Int'l Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2002). IEEE Comput. Soc. pp. 213-220
5. A. Bosworth. Developing Web Services. Proceedings 17th International Conference on Data Engineering. IEEE Comput. Soc, pp.477-81. Los Alamitos, CA, USA, 2001.
6. A. David , Chappell and Tyler Jewell. Java Web Services. *O'Reilly*.
7. D. Ferraiolo, J. Cugini, and D. Kuhn (1995). Role-Based Access Control (RBAC): Features and Motivations. In Computer Security Applications, pages 241–248.
8. S. Hada, M. Kudo. XML Access Control Language: Provisional Authorization for XML Documents. Tokyo Research Laboratory,IBM Research, 2003. (http://www.alphaworks.ibm.com/aw.nsf/download/xmlsecuritysuite)
9. H. Kreger. Web Services Conceptual Architecture. IBM Software Group May 2001
10. RFC 2459. Internet X.509 Public Key Infrastructure Certificate and CRL Profile.
11. XACML: eXtensible Access Control Markup Language. http://www.oasis-open.org/committees/xacml/repository/
12. AA.VV. UDDI technical white paper Role Uddi.Org, september 2000.

Supporting Location-Aware Distributed Applications on Mobile Devices

Cristiano di Flora[1], Massimo Ficco[2], and Stefano Russo[1,2]

[1] Computer Science Department
"Federico II" University of Naples
Via Claudio 21, 80125 - Napoli, Italy
{diflora,stefano.russo}@unina.it

[2] Laboratorio Nazionale per l'Informatica e la Telematica Multimediali CINI - ITEM
Via Diocleziano 328, 80124 - Napoli, Italy
massimo.ficco@napoli.consorzio-cini.it

Abstract. This work proposes to extend the Java APIs for Bluetooth (JSR82) in order to provide the Location API (JSR179) with a source of indoor-location information. The proposed extension relies on a specific indoor positioning technique to track current location. The adopted technique uses the Received Signal Strength Indicator (RSSI) as a good room-fingerprint. We evaluate the effectiveness of the approach by examining preliminary experimental results obtained from our first system prototype.

1 Introduction

The Wireless World Research Forum (WWRF) has identified device diversity and ambient-awareness as two key properties of applications and services in future mobile systems [1]. According to the WWRF, emerging applications for mobile devices will be adaptive, personalized, and context-aware. Indoor applications are specially concerned with ambient-awareness. Indeed, most of these applications adapt their behavior [2] to the physical location of users, and of other entities (i.e. services and devices) [3] as well. Moreover, they will be delivered to different end-devices (phones, computers, home appliances) over diverse wireless networks. Despite this diversity, many mobile devices (e.g. smart-phones and PDAs) now support the Mobile Information Device Profile of the Java 2 Micro Edition (J2ME) platform. This platform provides a common yet flexible computing and communication environment that can be fitted for (and shared by) devices having different capabilities. Many modern mobile terminals are also equipped with wireless connectivity devices, such as IEEE 802.11 or Bluetooth adapters. Therefore, the Java Community Process has recently finalized two Java Specification Requests (JSRs) in order to cope with location-awareness (JSR-179) [5] and Bluetooth communication (JSR82) [6] in Connected Limited Device Configuration (CLDC).

Several indoor-positioning techniques, including the one we presented in [7], exploit the Received Signal Strength Indicator (RSSI) to determine the zone that the mobile device is operating in. Though Bluetooth's Host Configuration Interface allows the RSSI to be read, the JSR82 API does not provide such functionality; hence it is not suitable for RSSI-based approaches. In this paper we propose to extend the JSR82 API in order to provide the Location API with a source of local information. We also show how to implement an RSSI-based locationing technique in compliance with the JSR179 API. Our solution is based on the insertion of a new component, called `RSSI_Provider`, into the JSR82 API. This is in charge of producing information about signal strength and link quality, which is needed in order to build `Location` objects. We bridged the two APIs by implementing a specific `LocationProvider` class (i.e. a location-providing module, generating `Locations`, as stated in JSR179), which exploits the `RSSI_Provider` to generate `Locations`. The proposed solution can suit a wide variety of mobile devices, since it requires no additional positioning device but a Bluetooth adapter. Based on the designed extensions, we propose an implementation of the `LocationProvider`, which relies on a specific RSSI-based indoor positioning technique [7].

The rest of the paper is organized as follows. In Section 2 we describe several approaches to develop Location-Aware applications for mobile devices. Section 3 casts some light on the Bluetooth and Location APIs. In Section 4 we present our solution with some comments about the current prototype implementation. In Section 5 we provide preliminary experimental results, and present some conclusions.

2 Developing Location-Aware Distributed Applications

In the last years a great deal of research has been conducted to realize systems and technologies for automatic location-sensing of either people or devices. Each existing approach solves a different problem or supports different applications. The work in [3] proposed a taxonomy for location-sensing systems. This taxonomy is based on several parameters, such as the physical phenomena used for determining location, the form factor of the sensing apparatus, power requirements, infrastructure, and resolution in time and space. A mobile user's location can be determined using several techniques, which are comprehensively described in [8]. Several positioning systems that use a triangulation technique have been proposed [9,10,11,12]; such systems provide location information by measuring either the distance between known points or the relative angle with respect to known referring poles. In [7,13] location is calculated as a function of the proximity to a known set of points. The mentioned works rely on different technologies, such as Bluetooth, IEEE 802.11 RF, GPS, and ad-hoc solutions. They are conceptually independent of the adopted technologies and techniques; unfortunately, to the best our knowledge, they lack a common high-level software application programming interface for location sensing. Thus using these solutions may result in ever heterogeneous applications.

As for location-aware computing, no single location-sensing technology is likely to become dominant, since each technology can either satisfy different requirements or suit different devices. Since several technologies will coexist into future mobile systems, such systems will need a high-level software application programming interface for technology-independent location sensing [4]. Several organizations have recently proposed location APIs to leverage the interoperability of outdoor positioning systems. The Open Mobile Alliance (OMA) Location Working Group (LOC), which continues the work originated in the former Location Interoperability Forum (LIF) [15], is developing specifications to ensure interoperability of Mobile Location Services on an end-to-end basis. This working group has defined a location services solution, which allows users and applications to obtain location information from the wireless networks independently of the air interface and positioning method. The Finnish Federation for Communications and Teleinformatics (FiCom) has built a national location API to enable location-information exchange between network operators, and between operators and service providers as well [14]. FICOM's API is compliant with the LIF location services specifications.

3 The Bluetooth and Location APIs for J2ME CLDC Profile

The JSR179 [5] defines a J2ME optional package to enable location-aware applications for mobile devices. The functionality addressed by this specification can be classified into the following two main categories: i) obtaining information about location and orientation of the mobile device; and ii) accessing a landmark database stored on the device. Each functionality exploits specific objects as information containers; indeed, the `Location` class represents the standard set of basic device-location information, whereas the `Landmark` class represents known locations (i.e. the places that mobile users can go through).

The `LocationProvider` class represents a module that is able to determine the location of the terminal. This may be implemented by using existing location methods, including satellite based methods like GPS, and short-range positioning methods like Bluetooth Local Positioning.

The JSR82 [6] aims to define a standard set of APIs that will enable an open software-development environment for Bluetooth-based devices. The main functionality provided by this specification encompasses discovery, communication, and device management. As for the discovery API, it is used to register services, as well as to discover devices and services. The communication API is used to establish connections between devices, including RFCOMM, L2CAP, and OBEX connections. The device management API allows to manage and control the communication connections, and to provide security for these activities as well.

4 Supporting RSSI-Based Locationing on J2ME Devices

In this work we adopt an indoor-positioning technique which is suitable for detecting the symbolic position [3] of mobile users within a set of buildings. We believe that a good understanding of the underlying positioning technique can also clarify the logic which is behind the proposed extension. Hence, we briefly describe this technique before analyzing in detail the proposed extension to the JSR82. The interested reader may refer to [7] for further details on the adopted approach. According to this technique, each room of the building is represented by a symbolic location, called "zone"; we assume the presence of one sensor for each zone. The topology of the considered environment is mapped through the *Sensor-Coordinates* table, which associates each Bluetooth sensor to a specific room, and the *Sensor-Neighbors* table, which logically links each sensor to the sensors of adjacent rooms.

The adopted technique uses the Received Signal Strength Indicator (RSSI) as a good fingerprint for the room the mobile device is in. It is worth noting that, although our representation of location implies an additional layer of indirection to retrieve the geometric information, it has some great advantages. First, storage and retrieval of symbolic location data can be more suitable to location applications than geometric location data. Second, a hierarchical (e.g., building-floor-room) symbolic data model can ease the integration of multiple wireless technologies within the same application. The third advantage is that the symbolic model can be used to predict user location, for it helps building secondary models of location information such as individual mobility patterns.

In the rest of this section we show how the adopted positioning technique can be mapped onto the JSR179 API. Moreover, we show how we extended the JSR82 API to bridge it to the JSR179's `LocationProviders`, and provide some comments about the current implementation.

4.1 Representing Rooms as JSR179 Locations

According to the zoning approach, each `Location` has a one-to-one relationship with a specific room of the building. Within the JSR179 specifications, `Location` objects represent the up-to-date location in terms of timestamped coordinates, accuracy, speed, course, and information about the positioning method used for the location, plus an optional textual address. Thus, it is crucial that our technique be mapped onto JSR179 semantics. As for the `QualifiedCoordinates`, we associate each room to a specific set of latitude, longitude, and altitude parameters. `Landmarks` and the already mentioned tables represent the topology of the considered environment. As for `Landmarks`, we describe each known `Location` (i.e., each room) by means of a name that identifies the room to the end user (e.g., the Contemporary Art gallery within an exhibition). We manage each table by means of a specific component. The `SensorLocator` is in charge of managing the *Sensor-Coordinates* table; hence it associates each zone (i.e. each Landmark) to a specific Bluetooth sensor. The `SensorConnector` manages the *Sensor-Neighbors* table; hence it represents the interconnections between rooms.

4.2 Enabling the RSSI-Based Locationing Technique

In order to enable the mentioned technique, we added a new class to the JSR82, namely the `RSSI_Provider`, which allows to measure the value of the RSSI, as well as to measure the Bluetooth connection's Link-Quality (i.e., the measurement of signal quality in terms of bit error rate (BER)). Both the mentioned measurements refer to the communication between the device itself and another Bluetooth device (i.e., a sensor).

Figure 1 shows the retrieval of user's location by means of the described extensions. It is worth noting that the `LocationEstimator` component exploits the `RSSI_Provider` to discover the zone where the mobile device is. It specifically analyzes the *Sensor-Neighbors* table by means of the `SensorConnector` component, and reads the RSSI value for each neighbored sensor. Upon the identification of the nearest sensor (i.e., the identification of the current room), the `LocationEstimator` can i) retrieve its spatial coordinates by means of the `SensorLocator` component, and ii) return them to the `LocationProvider`.

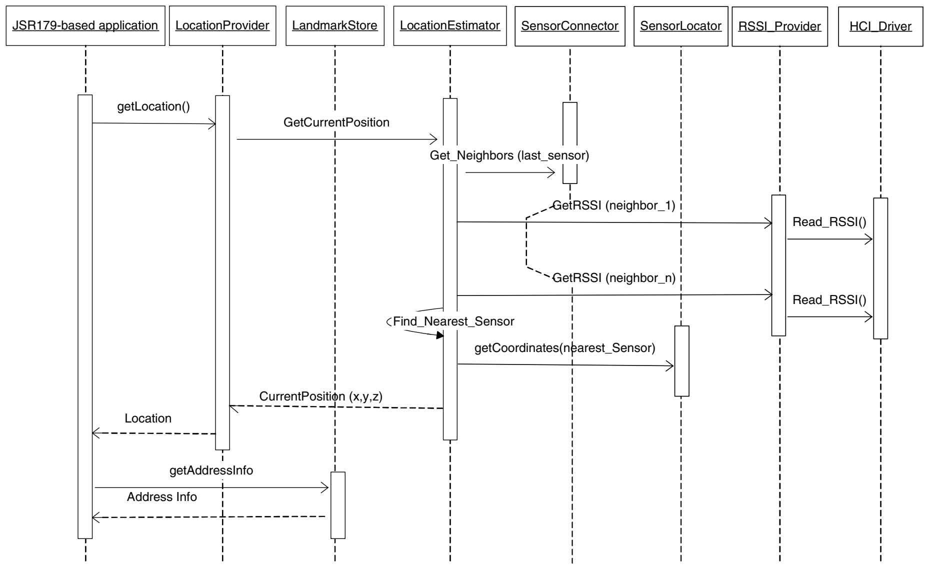


Fig. 1. Retrieval of a mobile user's location

Implementation Details. We implemented the `RSSI_Provider` class as an extension to the JBlueZ library. JBlueZ is an implementation of the JSR82 specifications, which relies on the official Linux Bluetooth protocol stack, namely BlueZ [16]. Since BlueZ is implemented as library of native procedures written in C, the `RSSI_Provider` uses the Java Native Interface (JNI) to retrieve RSSI and quality information through the BlueZ's Host COnfiguration Interface. As for the Location API, we developed a JSR179 implementation skeleton consisting of a `LandmarkStore` and a `LocationProvider`.

5 Experimental Results and Future Work

We tested the solution on Compaq iPAQ 3970 PDAs running the Familiar 0.7.0 Linux distribution. As for sensors, we used ANYCOM Bluetooth dongles configured to accept connections from mobile devices.

The results described herein were obtained by measuring the RSSI multiple times while moving through a simple sequence of three adjacent rooms. We put one sensor in the middle of each room; in other words, each zone represents a single room. Figure 2 depicts the topology and the preliminary results of the experiments we conducted on the first system prototype. As Figure 2 shows, during these preliminary experiments, the mobile device moves from Zone 1 to Zone 3, passing through Zone 2. The current zone is discovered by comparing the RSSI read from the sensor of the current zone and the one of the adjacent zones. Figure 2 shows also how the strength of the signal coming from each sensor varies according to device movements. Preliminary experiments showed that the time required to detect the movement from a certain room to another one is not

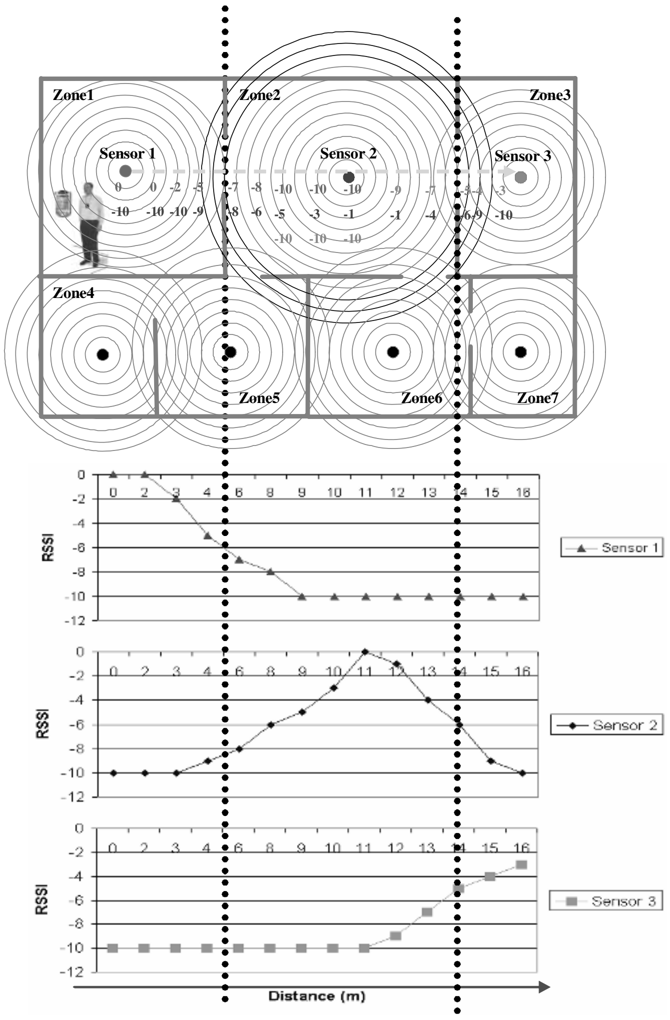


Fig. 2. Signal strength while moving between different rooms

significantly influenced by the distance from sensors, and it is smaller than $100ms$, which is a reasonable time if compared to the speed of a walking user. Hence the proposed approach can cope with the change in user's location while providing location-aware applications with a JSR179-compliant interface.

It is worth pointing out that we were interested in identifying the room that the device is moving in; thus the accuracy information in `Location` objects is related to the coordinates-error probability (i.e. associating a device to a wrong room). Since an effective measurement of such an error probability goes beyond the scope of this work, we assumed here the coordinates to have a constant accuracy. Future work will aim to address the following issues: i) evaluating the accuracy and device-speed limitations of the implemented mechanism; *ii*) defining an approach to determine the best location of each sensor in a certain topology; *iii*) evaluating and improving the robustness of the provided positioning services; and *iv*) experimenting the proposed approach over a different wireless-communication technology than Bluetooth (e.g., WiFi).

Acknowledgments

This work has been partially supported by the Italian Ministry for Education, University and Research (MIUR) in the framework of the FIRB Project "Middleware for advanced services over large-scale, wired-wireless distributed systems (WEB-MINDS), and by Regione Campania in the framework of "Centro di Competenza Regionale ICT").

References

1. Wireless World Research Forum. *Book of Visions 2001.* http://www.wireless-world-research.org/
2. K. Raatikainen, H.B. Christensen and T. Nakajima. Application Requirements for Middleware for Mobile and Pervasive Systems. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(4):16–24, October 2002.
3. J. Hightower, and G. Borriello. Location Systems for Ubiquitous Computing. *IEEE Computer*, 34(8):57–66, August 2001.
4. C.A. Patterson, R.R. Muntz and C.M. Pancake. Challenges in Location-Aware Computing. *IEEE Pervasive Computing*, 2(2):80–89, June 2003.
5. Java Community Process. Location API for J2ME Specification 1.0 Final Release. September 2003.
6. Java Community Process. Java APIs for Bluetooth Specification 1.0 Final Release. March 2002.
7. F. Cornevilli, D. Cotroneo, M. Ficco, S. Russo, and V. Vecchio. Implementing Positioning Services Over an Ubiquitous Infrastructure. accepted for publication into the *Proceedings of* 2^{nd} *IEEE Workshop on Software Technologies for Embedded and Ubiquitous Computing Systems (WSTFEUS '04)*, Vienna, Austria, May 2004.
8. J. Hightower, and G. Borriello. Location Sensing Techniques. *Technical report UW-CSE-01-07-01*, University of Washington, July 2001.
9. A. Harter et al. The Anatomy of a Context-Aware Application. *Proceedings of* 5^{th} *Annual International Conference on Mobile Computing and Networking (Mobicom 99)*, pp. 59–68, ACM Press, New York, 1999.
10. P. Bahl, and V.N. Padmanabham. RADAR: An In-Building RF-based User Location and tracking System. In *Proceedings of the IEEE Infocom 2000*, Vol.2, pp. 775–784, Tel-Aviv, Israel, March 2000.
11. G. Anastasi, R. Bandelloni, M. Conti, F. Delmastro, E. Gregori, and G. Mainetto. Experimenting an Indoor Bluetooth-Based Positioning Service. In Proceedings of the 23^{rd} International Conference on Distributed Computing Systems Workshops (ICDCSW'03), pp. 480–483, IEEE CS Press, 2003.
12. A. Kotanen, M. Hannikainen, H. Lappakoski, and T.D. Hamalainen. Experiments on Local Positioning with Bluetooth. In *Proceedings of the International Conference on Information Technology, Computers and Communication (ITCC03)*, pp. 297–303, IEEE CS Press, 2003.

13. Y. Fukuju, M. Minami, H. Morikawa and T. Aoyama. DOLPHIN: An Autonomous Indoor Positioning System in Ubiquitous Computing Enviroment. In *Proceedings of the IEEE Workshop on Software for Future Embedded Systems (WSTFES'03)*, pp. 53–56, Hakodate, Japan, May 2003.
14. FiCom Location API Working Group. FiCom Location API 2.0.0 Interface specification, 2002.
15. Location Inter-operability Forum. Mobile Location Protocol LIF TS 101 Specification, Version 3.0.0, June 2002.
16. Bluez: Official Linux Bluetooth protocol stack. *http://bluez.sourceforge.net/*

Grid Application Development on the Basis of Web Portal Technology[⋆]

Gábor Dózsa, Péter Kacsuk, and Csaba Németh

MTA SZTAKI, Laboratory of Parallel and Distributed Systems
H-1518 Budapest, Hungary
{dozsa,kacsuk,csnemeth}@sztaki.hu
http://www.lpds.sztaki.hu

Abstract. Providing Grid users with a widely accessible, homogeneous and easy-to-use graphical interface is the foremost aim of Grid-portal development. These portals, if designed and implemented in a proper and user-friendly way, might fuel the dissemination of Grid-technologies, hereby promoting the shift of Grid-usage from research into real life, industrial application, which is to happen in the foreseeable future, hopefully. Grid portals are quite frequently applied in order to utilize common user tasks like access control, data and application code transfers, job control and status monitoring but design, development and performance tuning of complex Grid or HPC applications are usually not supported directly by such portals. This paper introduces P-GRADE Portal being developed at MTA SZTAKI. It allows users to manage the whole life-cycle of building and executing complex applications in the Grid: editing workflows, submitting jobs relying on Grid-credentials and analyzing the monitored trace-data by means of visualization.

1 Introduction

Easy and convenient access of Grid systems is a foremost need for Grid end-users as well as for Grid application developers. Grid portals are the most promising environments to fulfill these requirements and hence we decided to create a Grid portal for our P-GRADE (Parallel Grid Run-time and Application Development Environment) system.

P-GRADE is an integrated graphical programming environment to develop and execute parallel applications on supercomputers, clusters and Grid systems [8][9]. P-GRADE supports the generation of either PVM or MPI code, creation and execution of Condor or Globus jobs. More advanced features include support for application migration, dynamic load-balancing, workflow definition and coordinated multi-job execution in the Grid. The aim of the portal is to make available these functionalities through the web. While P-GRADE in its original form requires the installation of the whole P-GRADE system on the client machines, the portal version needs only a web browser and all the P-GRADE portal services are provided by one or more P-GRADE portal servers that can be connected to various Grid systems.

[⋆] The work presented in this paper was partially supported by the following grants: EU-GridLab IST-2001-32133, IHM 4671/1/2003, Hungarian Scientific Research Fund (OTKA) No. T042459 and National Research and Technology Office, No. IKTA-00064/2003 projects.

We developed the portal using GridSphere, a Grid portal development framework [3]. The framework among other things saved us from implementing user management for the portal, by using its tag-library we could easily develop a coherent web interface, and enjoyed the way one can configure and manage portlets in GridSphere.

In general, a grid portal has to deal with three basic types of objects: users, resources and applications. The P-GRADE portal currently provides the following fundamental services.

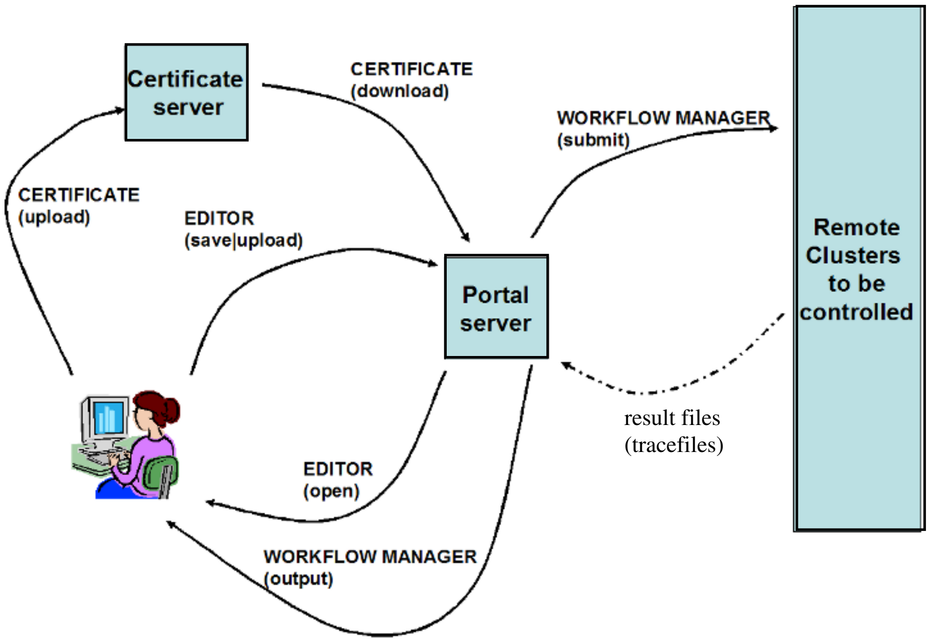

Certificate server
CERTIFICATE (download)
WORKFLOW MANAGER (submit)
CERTIFICATE (upload)
EDITOR (save|upload)
Portal server
Remote Clusters to be controlled
result files (tracefiles)
EDITOR (open)
WORKFLOW MANAGER (output)

Fig. 1. P-GRADE Portal Services

- *User management*: User accounts are managed by the help of the core GridSphere portlets. As an extension of the fundamental user management tasks, we have developed a new portlet for credential management. It aims at providing a convenient interface for the users to contact various MyProxy servers in order to get proxy certificates. The portlet can manage several proxy certificates for the same user and allows she to use any of them as required by particular computational resources.
- *Resource management*: Computational resources (i.e. clusters or supercomputers) on which applications can be executed from the portal are represented simply as a list of Globus-2 contact strings. The portal provides a default list (set up by the portal administrator) but each individual user can customize it by including additional resources to which the user has access or disabling some of them as she likes.
- *Application management*: There are two main aspects of application management: application development and execution control of existing applications. Although many portals put the focus on the execution control, we think that development of

new applications is rather important as well. Not many Grid applications exist today so the users (or application developers) need appropriate tools to make legacy applications Grid enabled or to use them as components of complex Grid applications. The P-GRADE portal provides the usual execution control mechanism (start/kill jobs, transfer input/output data) including status monitoring and animation. Furthermore, as a first step towards Grid application development support, the portal is equipped by a graphical workflow editor and a job performance visualization system. They are described in detail in the next section.

The common schema for a new user to start working with the P-GRADE Portal is the following. First, the user must upload a valid X.509 certificate to a certificate server (MyProxy) if she has not been uploaded it before. From the portal, she can then download a proxy certificate that enables her to run jobs in the Grid. The next step is to invoke the workflow editor that runs on the user's machine as a Java WebStart application. By the help of this editor, complex workflows can be defined by graphical means. Workflows can be stored (uploaded) on the portal server machine by the help of the editor to store them in a persistent (i.e. client machine independent) way. After finishing the design, the user can submit the workflow to the GT-2 Grid for execution. During the execution, the portal provides job status information as well as execution behaviour visualisations for each active workflow and their component jobs. Fig. 1 depicts such basic development steps supported in the current version of the P-GRADE Portal.

In the rest of the paper we describe the application development and execution control capabilities of the P-GRADE portal in brief - on the basis of a meteorology application as a case study - and outline our future plans to improve the usability of our portal.

2 Workflow Editing and Execution

A workflow is a set of consecutive and parallel jobs which are cooperating in the execution of a complex application. One job's output may provide input for the next. In the literature workflow is also referenced as application flow. The workflow concept is a widely accepted and convenient approach to construct complex large coarse grain applications - that fit well to be executed in a Grid - on the basis of existing programs as components.

In the P-GRADE portal, a graphical workflow editor is provided for the user to construct workflow applications.

For illustration purpose we use a meteorological application [4] called MEANDER developed by the Hungarian Meteorological Service. The aim of MEANDER is to analyze and predict in the ultra short-range (up to 6 hours) those weather phenomena which might be dangerous for life and property. Typically such events are snow-storms, freezing rain, fog, convictive storms, wind gusts, hail storms and flash floods. The complete MEANDER package consists of more than ten different algorithms from which we have selected four to compose a workflow application for demonstration purpose. Each calculation algorithm is computation intensive and implemented as a parallel program written in mixed C/C++ and FORTRAN.

Fig. 2 shows the workflow editor - which is implemented as a Java WebStart application - with the MEANDER meteorology application loaded in. The editor can be

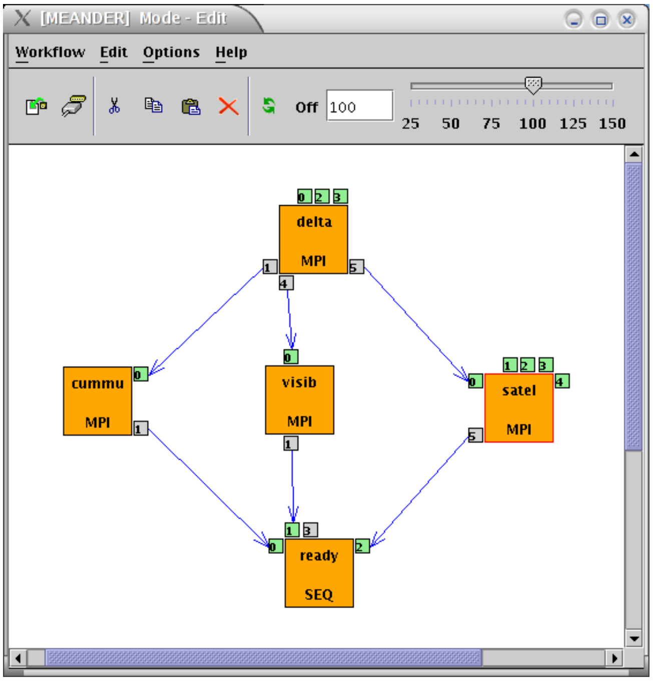


Fig. 2. Workflow editor of the P-GRADE portal

launched from the user's Web browser after logging into our portal. The important aspect of creating a graph is the possibility of composing the workflow application from existing code components. These components can be sequential programs (e.g. Ready in Fig. 2) or MPI programs (e.g. Satel in Fig. 2). This flexibility enables the application developer to create computation-intensive applications where several components are parallel programs to be run on supercomputers or clusters of the Grid.

Small rectangles (labelled by numbers) around nodes represent data files (of types input or output) of the corresponding job and directed arcs interconnect pairs of input and output files if an input file serves as an output for another job. In other words, arcs denote the necessary file transfers among jobs. A job can be started when all the necessary input files are available and transferred to the site where the job is allocated for execution. Naturally, independent jobs can be executed in parallel. Managing the file-transfers (both executables and data files) and recognition of feasibility to start the execution of a particular job is the task of our workflow manager that extends the capabilities of the Condor DAGMan [10] system serving as a Grid level job manager for the portal.

The P-GRADE workflow manager takes care of the - possible - parallel execution of these components. The graph depicted in Fig. 2 consists of four MPI jobs (nodes) corresponding to four different parallel algorithms of the MEANDER ultra-short range weather prediction package (*Delta*, *Cummu*, *Visib* and *Satel*) and a sequential job that collects the final results and presents them to the user as a kind of meteorological map (*Ready*).

This distinction among job types is necessary because the job manager on the selected grid site should be able to support the corresponding parallel execution mode, and the workflow manager is responsible for the handling of various job types by generating the appropriate submit files.

Besides the job type and the name of the executable, the user can specify the necessary arguments and the hardware/software requirements (architecture, operating system, minimal memory and disk size, number of processors, etc.) for each job. To specify the resource requirements, the application developer can currently use either the Condor resource specification syntax and semantics for Condor based grids or the explicit declaration of a grid site where the job is to be executed for Globus based grids.

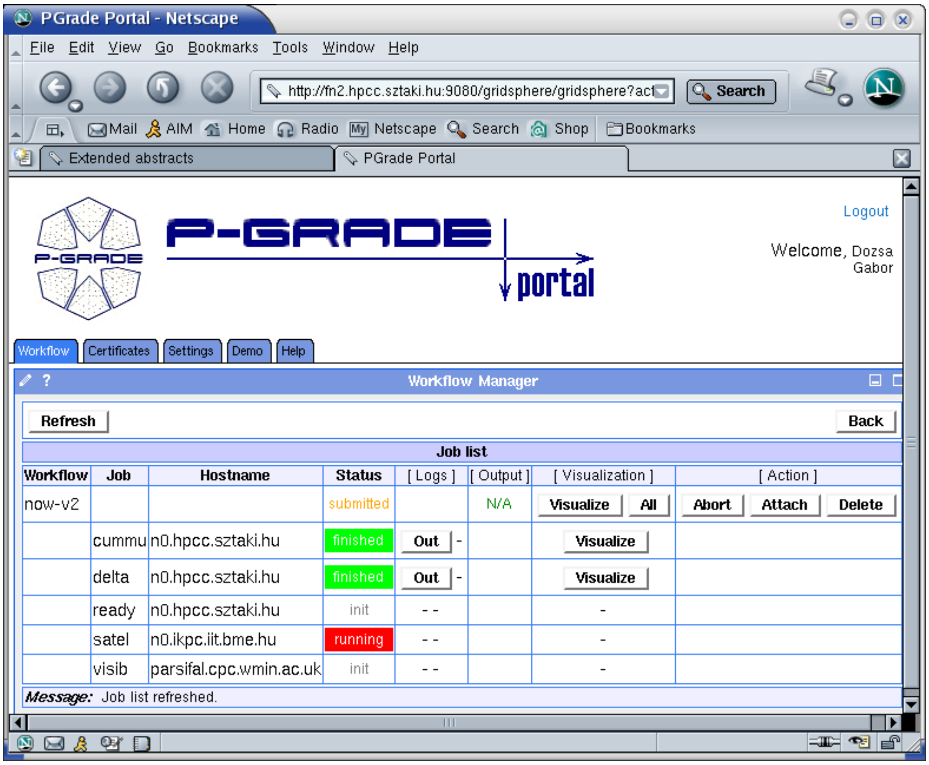
In order to define the necessary file operations of the workflow execution, the user should define the attributes of the file symbols (ports of the workflow graph) and file transfer channels (arcs of the workflow graph). The main attributes of the file symbols are as follows:

- file name
- type

The type can be permanent or temporary. Permanent files should be preserved during the workflow execution but temporary files can be removed immediately when the job using it (as input file) has been finished. It is the task of the workflow manager to transfer the input files to the selected site where the corresponding job will run. The transfer can be done in two ways. The off-line transfer mode means that the whole file should be transferred to the site before the job is started. The on-line transfer mode enables the producer job and the consumer job of the file to run in parallel in a pipeline fashion. When a part of the file is produced the workflow manager will transfer it to the consumer's site. However, this working mode obviously assumes a restricted usage of the file both at the producer and consumer sites and hence, it should be specified by the user that the producer and consumer meet these special conditions. In the current implementation only the off-line transfer mode is supported.

Once the workflow is specified, the necessary executables and input files should be uploaded to the P-GRADE portal server - this can be initiated from the workflow editor - and then the server will take care of the complete execution of the workflow and the transfer of results back to the client.

Fig. 3 shows the job status page (within the Netscape browser) of a running instance of the MEANDER application. Beside displaying the current statuses of all workflow jobs, this page also serves for controlling the execution of the workflow and also viewing and downloading standard output,error and final result files of jobs. Final results files can be downloaded in a compressed tarball form to the client machine after the execution having finished while standard output and error files are refreshed regularly and can be visited at any time during the execution as well.


PGrade Portal - Netscape
File Edit View Go Bookmarks Tools Window Help
http://fn2.hpcc.sztaki.hu:9080/gridsphere/gridsphere?act
Search
Mail AIM Home Radio Netscape Search Shop Bookmarks
Extended abstracts
PGrade Portal
P-GRADE
P-GRADE portal
Logout
Welcome, Dozsa Gabor
Workflow Certificates Settings Demo Help
Workflow Manager
Refresh
Back

Job list							
Workflow	**Job**	**Hostname**	**Status**	[Logs]	[Output]	[Visualization]	[Action]
now-v2			submitted		N/A	Visualize All	Abort Attach Delete
	cummu	n0.hpcc.sztaki.hu	finished	Out -		Visualize	
	delta	n0.hpcc.sztaki.hu	finished	Out -		Visualize	
	ready	n0.hpcc.sztaki.hu	init	- -		-	
	satel	n0.ikpc.iit.bme.hu	running	- -		-	
	visib	parsifal.cpc.wmin.ac.uk	init	- -		-	

Message: Job list refreshed.

Fig. 3. Status page of the active MEANDER workflow

3 Performance Monitoring and Visualization

Besides creating and managing workflows the P-GRADE portal enables on-line monitoring and visualization of the workflow execution. For monitoring jobs we use the Mercury [5] monitoring tool developed in our laboratory within the framework of the GridLab EU project . Its architecture complies with the Grid Monitoring Architecture proposed by the Global Grid Forum. Producers send measurement data to consumers upon request. For job monitoring, when submitting a job we subscribe to the corresponding metric. As progress unfolds, Mercury will send trace-data back to the TraceFile Monitor component in the portal, which collects trace-files and passes them to the visualizer applet provided that the Mercury monitoring service is available on the executing Grid resource and the application is instrumented with the necessary trace generation function calls.

Basically, there are two ways to obtain such an instrumented parallel application. One can use the P-GRADE program development environment to create the parallel application by graphical means. Probably this is the most convenient way since, P-GRADE can generate automatically the proper instrumented MPI executables to be used as a component in the portal.

The other way is to insert all the necessary instrumentation function call into the MPI source code either by hand or by the help of our semi-automatic instrumentation script. The latter can be used if the MPI program complies with some basic restrictions like relying only on the default communicator.

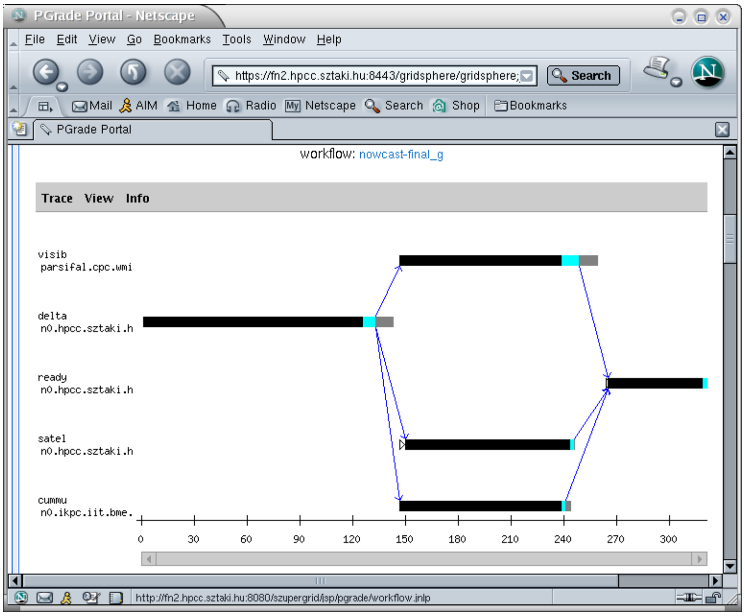

PGrade Portal - Netscape
File Edit View Go Bookmarks Tools Window Help
https://fn2.hpcc.sztaki.hu:8443/gridsphere/gridsphere;
Search
Mail AIM Home Radio Netscape Search Shop Bookmarks
PGrade Portal
workflow: nowcast-final_g
Trace View Info
visib
parsifal.cpc.umi
delta
n0.hpcc.sztaki.h
ready
n0.hpcc.sztaki.h
satel
n0.hpcc.sztaki.h
cummu
n0.ikpc.iit.bme.
0
30
60
90
120
150
180
210
240
270
300
http://fn2.hpcc.sztaki.hu:8080/szupergrid/jsp/pgrade/workflow.jnlp

Fig. 4. Space-time diagram of the MEANDER workflow

There is a three level visualisation facility in the P-GRADE portal. First, a status window is available providing information for each workflow about the component jobs, their executing Grid sites and their status as illustrated in Fig. 3. The workflow level execution visualisation window shown in Fig. 4 graphically represents the progress of the workflow execution. In the workflow space-time diagram, the horizontal bar represents the progress of each component job in time (see the time axis at the bottom of the diagram) and the arrows among bars represent the file operations performed to make accessible the output file of a job as an input of another one. Finally, a similar window can be opened for any parallel component job. Interpretation of the same diagram elements is a bit different in this case (like job *Cummu*, see Fig. 5). Here the horizontal bars represent the progress of each process comprising the parallel job whereas arrows between bars represent (MPI) message transfers among the processes [11].

4 Related and Future Works

A Grid portal is usually deployed as an interface for a particular Grid testbed. This is the main reason behind the idea of portal frameworks like GridSphere, i.e. it is expected to

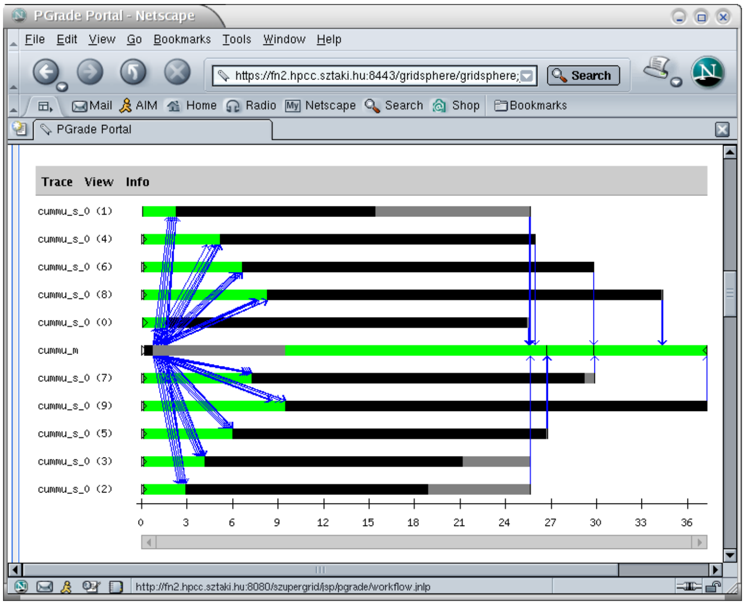


Fig. 5. Space-time diagram of the 'cummu' parallel job from the MEANDER workflow

implement a specific portal for each different Grid infrastructure. Our approach differs from this concept in the sense that we would like to provide a general purpose Grid portal equipped with all the necessary tools to support Grid application development and execution in an integrated way. As a first step towards this direction, our portal can harness the computational power of any Globus-2 type Grid (e.g. Hungarian SuperGrid [12] and GridLab [16] testbed) and it provides a convenient graphical workflow editor to compose large coarse grain Grid applications. As our primary targets are HPC applications, we provide integrated graphical support also for performance monitoring and visualisation.

To pursue our ultimate goal further, we plan to connect our portal to other types (not pure GT2 type) of Grids like Hungarian ClusterGrid [14] and NorduGrid [15]. We also would like to further develop our workflow manager in order not to burden the user with assigning jobs to computing resources instead, most promising resources should be selected by the workflow manager automatically relying on a Grid information system. With respect to the application development support we plan to extend the capabilities of the workflow editor in order to support also parallelization of existing sequential applications on the basis of the GRED editor approach [9]. In this way we will cover both the composition of Grid applications (workflows) and the parallelization of existing sequential computation intensive programs (i.e. making them Grid enabled HPC applications). Furthermore, we plan to adapt some advance tools exist in our original P-GRADE environment to the P-GRADE portal, like task migration and fault tolerant execution of message-passing applications.

With respect to our workflow editor/manager, several other similar systems exist like Unicore [13], Triana [17] and Pegasus [1]. None of them puts the focus on HPC application as our portal does by providing explicit support for MPI applications, performance visualisation and parallelization (in the near future). On the other hand, some of the other workflow systems (most notably Pegasus) can support parameter study like applications and automatic generation of complex large workflows that is currently out of the scope of the P-GRADE portal.

Executing the MEANDER meteorology application workflow through the P-GRADE portal has already been successfully presented at Supercomputing Conference and Exhibition 2003 in Phoenix/AZ (U.S.A.) [6], and at IEEE International Conference on Cluster Computing in Hong Kong/China, 2003 [7].

References

1. Ewa Deelman, et al: Mapping Abstract Complex Workflows onto Grid Environments, Journal of Grid Computing, Vol. 1, no. 1, 2003, pp. 25-39
2. J. Novotny: The Grid Portal Development Kit. IEEE Concurrency and Practice vol. 13, 2002
3. J. Novotny, M. Russell, O. Wehrens: GridSphere: A Portal Framework for Building Collaborations. 1st International Workshop on Middleware for Grid Computing, Rio de Janeiro, June 15, 2003
4. R. Lovas, et al.: Application of P-GRADE Development Environment in Meteorology. Proc. of DAPSYS'2002, Linz, pp. 30-37, 2002
5. Z. Balaton, G. Gombás: Resource and Job Monitoring in the Grid., Proc. of EuroPar'2003 Conference, Klagenfurt, Austria, pp. 404-411, 2003
6. Supercomputing 2003 in Phoenix/AZ (U.S.A.). http://www.sc-conference.org/sc2003/
7. IEEE International Conference on Cluster Computing in Hongkong/China. http://www.hipc.org/
8. P-GRADE Graphical Parallel Program Development Environment. http://www.lpds.sztaki.hu/pgrade
9. P. Kacsuk, G. Dózsa, R. Lovas: The GRADE Graphical Parallel Programming Environment. In the book: Parallel Program Development for Cluster Computing: Methodology, Tools and Integrated Environments (Chapter 10), Editors: P. Kacsuk, J.C. Cunha and S.C. Winter, pp. 231-247, Nova Science Publishers New York, 2001
10. James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke: Condor-G: A Computation Management Agent for Multi-Institutional Grids. Journal of Cluster Computing volume 5, pages 237-246, 2002
11. Z. Balaton, P. Kacsuk, and N. Podhorszki: Application Monitoring in the Grid with GRM and PROVE. Proc. of the Int. Conf. on Computational Science ICCS 2001, San Francisco, pp. 253-262, 2001
12. P. Kacsuk: Hungarian Supercomputing Grid. Proc. of ICCS'2002, Amsterdam. Springer Verlag, Part II, pp. 671-678, 2002
13. Unicore project, www.unicore.org
14. P. Stefán: The Hungarian ClusterGrid Project. Proc. of MIPRO'2003, Opatija, 2003
15. O.Smirnova et al: The NorduGrid Architecture And Middleware for Scientific Applications. ICCS 2003, LNCS 2657, p. 264. P.M.A. Sloot et al. (Eds.) Springer-Verlag Berlin Heidelberg 2003
16. GridLab project, www.gridlab.org
17. I. Taylor, et al: Grid Enabling Applications Using Triana, workshop onGrid Applications and Programming Tools, June 25, Seattle, 2003

A Distributed Divide and Conquer Skeleton

Juan R. González, Coromoto León, and Casiano Rodríguez

Dpto. Estadítica, I.O. y Computación
Universidad de La Laguna
E-38271 La Laguna, Tenerife, Spain
{jrgonzal,cleon,casiano}@ull.es

Abstract. The MaLLBa library provides skeletons to solve combinatorial optimization problems. Its main objective is to simplify the implementation of algorithms based on some commonly used techniques such as Branch and Bound, Dynamic Programming and Divide and Conquer.
This work is focused on the MaLLBa::DnC skeleton, which solves problems that fit in the Divide and Conquer paradigm. The user has to provide functions particularized to the problem he wants to solve. Given that functions the skeleton encapsulates all remaining work and allows the problem to be solved either in a sequential or parallel way.
In this work we will present a new MPI asynchronous peer-processor implementation of the MaLLBa::DnC skeleton where all processors are peers and behave the same way (except during the initialization phase) and where decisions are taken based only on local information. Results on a Linux cluster of PC for matrix and huge integer multiplication are presented.

1 Introduction

The Divide and Conquer technique is a general method to solve problems. Algorithms based in this technique divide the original problem in smaller subproblems, solve them and combine the obtained sub-solutions to get the solution of the original problem. The MaLLBa::DnC skeleton requires from the user the implementation of a C++ class `Problem` defining the problem data structures, a class `Solution` to represent the result and a class `SubProblem` to specify subproblems. In some cases, an additional `Auxiliar` class is needed to represent the subproblems which do not have exactly the same structure of the original one.

It is in the `SubProblem` class where the user has to provide the methods `easy()`, `solve()` and `divide()`. The `easy()` function must check if a problem is simple enough to apply the simple resolution method `solve()`. The `divide()` function must implement an algorithm to divide the original problem into smaller problems with the same structure as the original one.

The class `Solution` has to provide an algorithm to put together partial solutions in order to obtain the solution of a larger problem through the `combine()` function.

MaLLBa::DnC provides a sequential resolution pattern and a message-passing master-slave resolution pattern for distributed memory machines [3]. This work presents a new fully distributed parallel skeleton that provides the same user interface and consequently is compatible with already implemented codes. The new algorithm is a MPI [7] asynchronous peer-processor implementation where all the processors are peers and behave the same way (except during the initialization phase) and where decisions are taken based on local information.

There are several implementations of general purpose skeletons using Object Oriented paradigms [2], [6]. Regarding Divide-and-Conquer we can mention the following ones: Cilk [4] based on the C language and Satin [5] codified in Java. As far as we know there are no tools integrating more than one exact technique. An innovation of our approach is that it allows the combined and nested use of all MaLLBa skeletons [1].

The contents of the paper are as follows: section 2 describes the data structures used to represent the search space and the parallel MPI skeleton. Computational results are discussed in section 3. Finally, the conclusions and future prospectives are commented in section 4.

2 The Algorithm of the Skeleton

The algorithm has been implemented using Object Oriented and Message Passing techniques. Figure 1 graphically displays the data structure used to represent the tree space. It is a tree of subproblems (`sp`) where each node has: a pointer to its father, the solution of the subproblem (`sol`) and the auxiliar variable (`aux`) for the exceptional cases when the subproblems have not the same structure than the original one. Additionally, a queue with the nodes pending of being explored is kept (`p` and `n`). The search tree is distributed among the processors. Registered in each node is the number of unsolved children and the number of children sent to other processors (remote children). Also an array of pointers to the solutions of the children nodes (`subsols`) is kept. When all the children nodes are solved, the solution of the actual node, `sol`, will be calculated combining the partial solutions stored in `subsols`. The solution is sent up to the father node in the tree and the partial solutions are disposed. Since the father of a node can be located in another processor, the rank of the processor owning the father is stored on each node.

The implementation of the algorithmic skeleton has been divided in two stages: An intialization phase and a resolution phase.

2.1 The Initialization Stage

The data required to tackle the resolution of the problem is placed into the processors. Initially only the master processor (*master*) has the original problem and proceeds to distribute it to the remaining processors. Next, it creates the node representing the initial subproblem and inserts it into its queue. All the processors perform monitor and work distribution tasks. The state of the set of processors intervening in the resolution is kept by *all* the processors. At the beginning, all the processors except the *master* are idle. Once this phase is finished, all the processors behave the same way.

2.2 Resolution Stage

The goal of this stage is to find the solution of the original problem. The majority of the skeleton tasks occur during this phase. The main aim of the design of the skeleton has

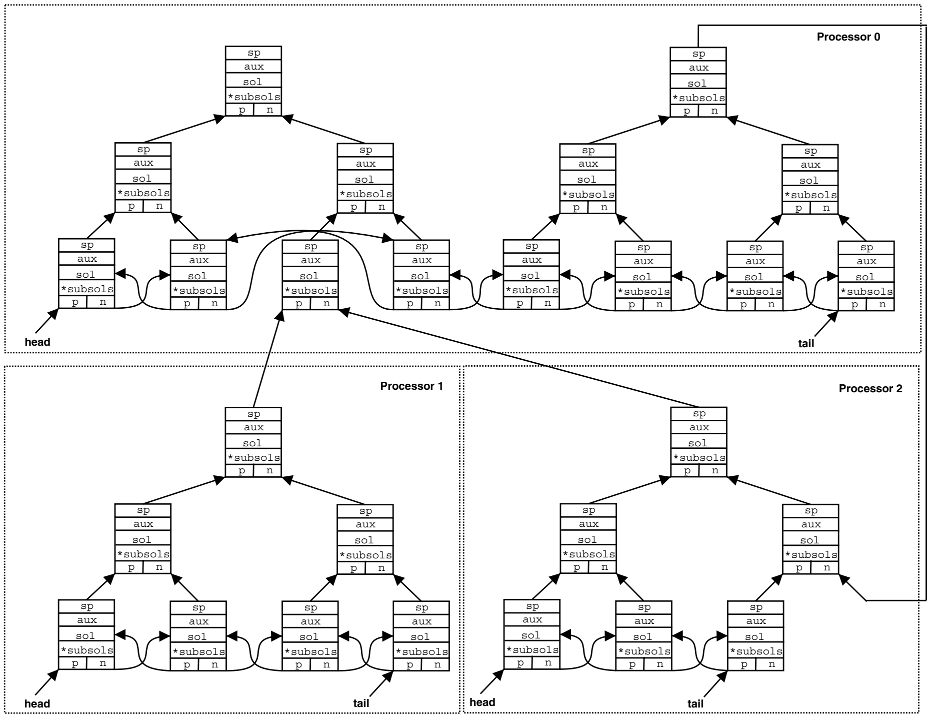


Fig. 1. Data structure for the tree space

been to achieve a *balanced distribution* of the work load. Furthermore, such arrangement is performed in a *fully distributed* manner using only asynchronous communication and information that is local to each processor, avoiding this way the use of a central control and synchronization barriers.

The developed control is based on a request-answer system. When a processor is idle it performs a request for work to the remaining processors. This request is answered by the remaining processors either with work or with a message indicating their inability to send work. Since the state of the processors has to be registered locally, each one sends its answer not only to the processor requesting work but also to the remaining others, informing of what the answer was (if work was sent or not) and to which processor it was. This allows to keep updated the state of *busy* or *idle* for each processor. According to this approach, the algorithm would finish when all the processors are marked idle. However, since request and answer messages may arrive in any arbitrary order, a problem arises: the answers may arrive before the requests or messages referring to different requests may be tangled. Additionally, it may also happen that a message is never received if the processors send a message corresponding to the resolution phase to a processor that has decided the others are idle.

To cope with these problems, a *turn of requests* is associated with each processor, assigning a rank or order number to the requests in such a way that it can be determined which messages correspond to what request. They also keep a *counter of pending ans-*

wers. To complete the frame, additional constraints are imposed to the issue of messages: (i) A processor that becomes idle, performs a single request until it becomes busy again. This avoids an excess of request messages at the end of the resolution phase. (ii) A processor does not perform any request while the number of pending answers to its previous request is not zero. Therefore, work is not requested until checking that there is no risk to acknowledge work corresponding to a previous request. (iii) No processor sends messages not related with this protocol while it is idle.

Though the former restrictions solve the aforementioned protocol problems, a new problem has been introduced: if a processor makes a work request and none sends work as answer, that processor, since it is not allowed to initiate new petitions, will remain idle until the end of the stage (*starvation*). This has been solved making the *busy* processors to check, when it is their turn to communicate, if there are idle processors without pending answers to their last petition and, if there is one and they have enough work, to force the initiation of a new turn of request for that idle processor. Since the processor(s) initiating the new turn are working, such messages will be received before any subsequent messages of work request produced by it (them), so there is no danger of a processor finishing the stage before that messages are received and therefore they will not be lost.

```
    ... // Resolution phase
 1  problemSolved = false;
 2  while (!problemSolved) {
 3    // Conditional communication
 4    if (time to communicate) {
 5      // Message reception
 6      while (pending packets) {
 7        inputPacket = packetComm.receive(SOLVING_TAG);
 8        switch (inputPacket.msgType) { // Messages types
 9          case NOT_WORKING_MSG:...
10          case CANT_SEND_WORK_MSG:...
11          case SEND_WORK_MSG_DONE:...
12          case SEND_WORK_MSG:...
13          case CHILD_SOL_MSG:...
14        } }
15      // Avoid starvation when a request for work was neglected
16    }
17    // Computing ...
18    // Work request ...
19    // Ending the resolution phase ...
20  } ...
```

Fig. 2. Resolution phase outline

The scheme in Figure 2 shows the five main tasks performed in this phase. The "conditional communication" part (lines 3-16) is in charge of the reception of all sort of messages and the work delivery. The communication is conditional since is not made per

```
 1 if (!dcQueue.empty()) { // Computing
 2   node = dcQueue.removeFromBack();
 3   sp = node.getSubProblem();
 4   if (sp.easy()) {
 5     sp.solve(node.getSolution());
 6     node = dcQueue.combineNode(pbm, node);
 7     if (node.getNumChildren() == 0) {
 8       if (node.getFather() == NULL) { // solution of original
 9         sol = node.getSolution();
10       }
11       else { // combination must continue in other processor
12         packetComm.send(node.getFatherProc(),
                           SOLVING_TAG, CHILD_SOL_MSG,
                           pack(node.getIdInFather(),
                                node.getFather(),
                                node.getSolution());
13   } } }
14   else {
15     sp.divide(pbm, subPbms[], node.getAuxiliar());
16     for (i = 1; i < subPbms[].size; i++) {
17       dcQueue.insertAtBack(dcNode.create(subpbms[i]), 0, i,
                                             node);
18 } } } ...
```

Fig. 3. Divide and Conquer computing tasks

iteration of the main loop but when the condition `time to communicate == TRUE` is held. The goal of this is to limit the time spent checking for messages, assuring the fulfilment of a minimum work between communications. The current implementation checks the time that has passed is larger than a given threshold. The threshold value has been established to a small value, since for larger values the work load balance gets poorer and the subsequent delay of the propagation of bounds leads to the undesired exploration of non promising nodes.

Inside the "message reception" loop (lines 6-14) the labels to handle the messages described in previous paragraphs are specified. Messages of type CHILD_SOL_MSG are used to communicate the solution of a subproblem to the receiver processor.

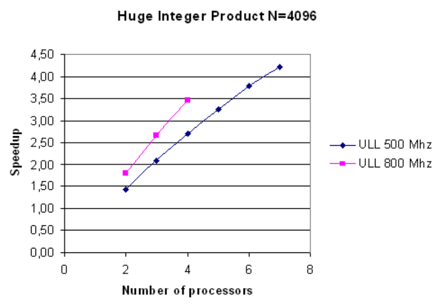
The "computing subproblems" part (line 17) is where the generic Divide and Conquer algorithm is implemented. Each time the processor runs out of work, one and only one "work request" (line 18) is performed. This request carries the beginning of a new petition turn. To determine the "end of the resolution phase" (line 19) there is a check that no processor is working and that there are no pending answers. Also, it is required that all the nodes in the tree have been removed. In such case the problem is solved and the optimal solution has ben found. The last condition is needed because the algorithm does not establish two separate phases for the division and combination processes, but both tasks are made coordinately. Therefore, if some local nodes have not been removed, there are one or more messages with a partial solution (CHILD_SOL_MSG) pending to be received, to combine them with the not deleted local nodes.

Figure 3 shows the computing task in more depth. First, the queue is checked and a node is removed if it is not empty. If the subproblem is easy (lines 4-13) the user method `solve()` is invoked to obtain the partial solution. Afterwards, the call to `combineNode()` will combine all the partial solutions that are available. The `combineNode()` method inserts the solution received as a parameter in the array of partial solutions `subsols` in the father node. If after adding this new partial solution the total number of sub-solutions in the father node is completed, all of them are combined using the user method `combine()`. Then, the father node is checked to know if the combination process can be applied again. The combination process stops in any node not having all its partial solutions or when the root node is reached. This node is returned by the `combineNode()` method. If the returned node is the root node (it has not father) and the number of child pending to solutions is zero, the problem is solved (lines 8-10). If the node has no unsolved children but it is an intermediate node, its father must be in another processor (lines 11-13). The solution is sent to this processor with a CHILD_SOL_MSG message and it continues with the combination process from this point. If the subproblem removed from the queue is not simple (lines 14-18), it is divided in new subproblems using the `divide()` method provided by the user. The new subproblems are inserted at the end of the queue. In order to traverse the tree in depth, the insert and remove actions in the queue are always made at the end. This allows a faster way to combine partial solutions and a more efficient use of memory.

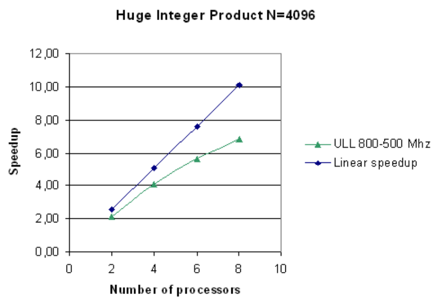
3 Computational Results

The experiments were performed instantiating the MaLLBa::DnC skeleton for some classic Divide and Conquer algorithms: sorting (quicksort and mergesort), convex hull, matrix multiplication, Fast Fourier Transform and huge integer product. Results were taken on a heterogeneous PC network, configured with: four 800 MHz AMD Duron processors and seven AMD-K6 3D 500 MHz processors, each one with 256 MBytes of memory and a 32 GBytes hard disk. The installed operating system was Debian Linux version 2.2.19 (herbert@gondolin), the C++ compiler used was GNU gcc version 2.95.4 and the message passing library was *mpich* version 1.2.0.

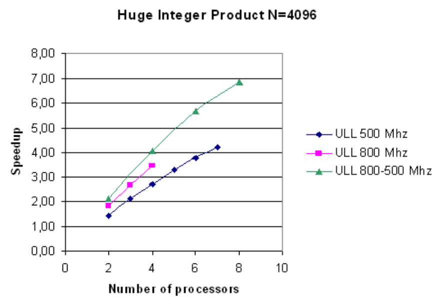
Figure 4 shows the speedups obtained for the huge integer product implementation with a problem of size 4,096. The parallel times were the average of five executions. The experiments labeled ‘ULL 500 Mhz’ and ‘ULL 800 Mhz’ were carried out on homogeneous set of machines of sizes seven and four respectively. Label ‘ULL 800-500 Mhz’ depict the experiment on a heterogeneous set of machines where half the machines (four) were at 800 MHz and the other half were at 500 MHz. The sequential execution for the ‘ULL 800-500 Mhz’ experiment was performed on a 500 MHz processor. To interpretate the ‘ULL 800-500 Mhz’ line take into account that the ratio between the sequential executions was 1.53. For eight processors the maximum speed up expected will be 10.12, that is, 1.53×4 (fast processors)$+4$ (slow processors), see Figure 4 (b). Comparing the three experiment depicted, Figure 4 (c), we conclude that the algorithm does not experiment any lost of performance due to the fact of being executed on a heterogeneous network. Figure 4 (d) represents the average number of visited nodes, for the experiment labelled ‘ULL 800-500 Mhz’. It is clear that an increase of the number of



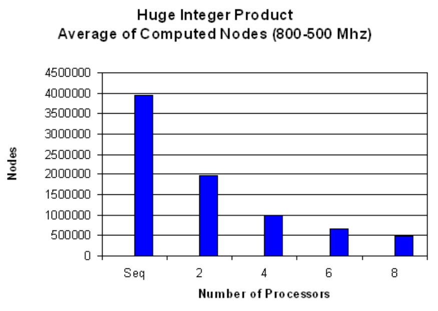
(a) Homogeneous case



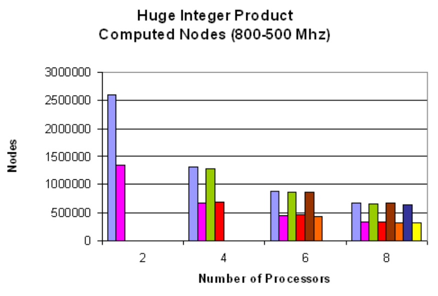
(b) Heterogeneous case



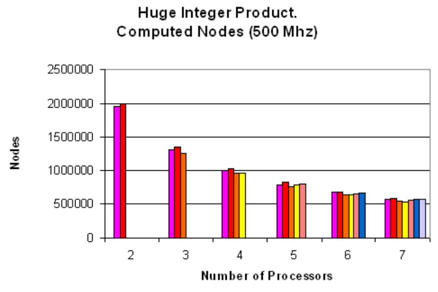
(c) Speedups



(d) Average number of visited nodes



(e) Heterogeneous case

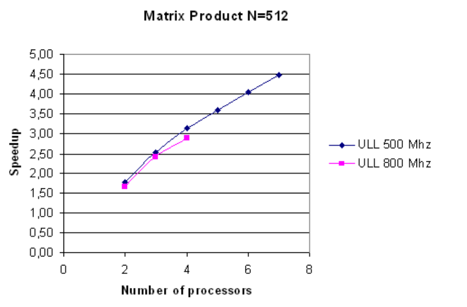


(f) Homogeneous case

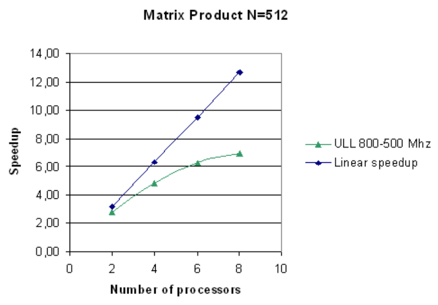
Fig. 4. Results for the Huge Integer Product

processors carries a reduction of the number of visited nodes. This evidences the good behaviour of the parallel algorithm.

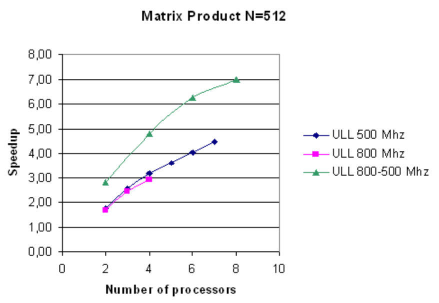
A parameter to study is the load balance among the different processors intervening in the execution of the algorithm. Figure 4 (e) shows the per processor average of the number of visited nodes for the five executions. Observe how the slow processors exa-



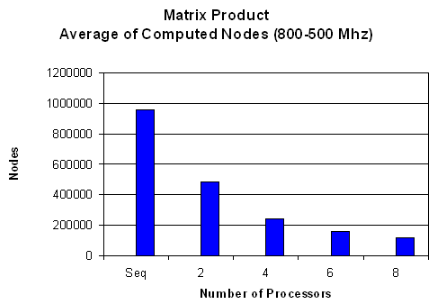
(a) Homogeneous case



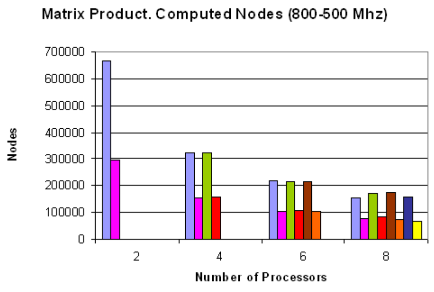
(b) Heterogeneous case



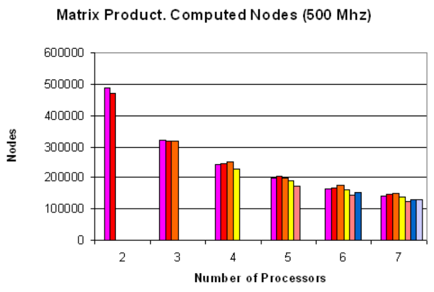
(c) Speedups



(d) Average number of visited nodes



(e) Heterogeneous case



(f) Homogeneous case

Fig. 5. Results for the Strassen's Matrix Multiplication

mine less nodes than the faster ones. It is interesting to compare these results with those appearing in Figure 4 (f) corresponding the homogeneous executions. Both pictures highlight the fulfilled fairness of the work load distribution.

Similar results are obtained for the implementation of the other algorithms. Figure 5 presentsthe same study for the Strassen's Matrix Product algorithm. Thogh the grain is finer than in the previous example, the performance behaviour is similar.

4 Conclusions

This work describes a parallel implementation of a skeleton for the Divide and Conquer technique using the Message Passing paradigm. The main contribution of the algorithm is the achievement of a balanced work load among the processors. Furthermore, such arrangement is accomplished in a fully distributed manner, using only asynchronous communication and information that is local to each processor. To this extent, the use of barriers and a central control has been avoided. The results obtained shows a good behaviour in the homogeneous and the heterogeneous case.

Ongoing work focuses on lighten the replication of information relative to the state of a certain neighbourhood. An OpenMP [8] based resolution pattern for shared memory machines is in the agenda.

Acknowledgements

This work has been partially supported by the EC (FEDER) and by the Spanish Ministry of Science and Technology inside the "Plan Nacional de I+D+I" with contract number TIC2002-04498-C05-05 and the Canary Government contract number COF2003-022.

References

1. E. Alba, F. Almeida, M. Blesa, J. Cabeza, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, C. León, J. Luna, L. Moreno, J. Petit, A. Rojas and F. Xhafa. MaLLBa: A Library of skeletons for combinatorial optimisation. In Proceedings of the International Euro-Par Conference. Paderborn, Germany, LNCS 2400, 927–932, 2002.
2. J. Anvik, S. MacDonald, D. Szafron, J. Schaeffer, S. Bromling and K. Tan. Generating Parallel Programs from the Wavefront Design Pattern. In Proceedings of the 7th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'02). Fort Lauderdale, Florida, 2002.
3. I. Dorta, C. León, C. Rodríguez and A. Rojas. Parallel Skeletons for Divide-and-Conquer and Branch-and-Bound Techniques. Proceedings of the 11th Euromicro Conference on Parallel, Distributed and Network Based Processing. Geneva, Italy, 292–298, 2003.
4. T. Kielmann, R. Nieuwpoort, H. Bal. Cilk-5.3 Reference Manual. Supercomputing Technologies Group. 2000.
5. T. Kielmann, R. Nieuwpoort, H. Bal. Satin: Efficient Parallel Divide-and-Conquer in Java. In Proceedings of the International Euro-Par Conference. 690–699, 2000.
6. H. Kuchen. A Skeleton Library. In Proceedings of the International Euro-Par Conference. 620–629, 2002.
7. M. Snir, S.W. Otto, S. Huss-Lederman, D.W. Walker, J.J. Dongarra. MPI: The Complete Reference. The MIT Press, 1996.
8. OpenMP Architecture Review Board. OpenMP C and C++ Application Program Interface. Version 1.0, http://www.openmp.org, 1998.

A Tool to Display Array Access Patterns in OpenMP Programs

Oscar R. Hernandez, Chunhua Liao, and Barbara M. Chapman

Computer Science Department
University of Houston
4800 Calhoun Rd.
Houston, TX 77204-3010
{oscar,liaoch,chapman}@cs.uh.edu

Abstract. A program analysis tool can play an important role in helping users understand and improve OpenMP codes. Array privatization is one of the most effective ways to improve the performance and scalability of OpenMP programs. In this paper we present an extension to the Open64 compiler and the Dragon tool, a program analysis tool built on top of this compiler, to enable them to collect and represent information on the manner in which threads access the elements of shared arrays at run time. This information can be useful to the programmer for restructuring their code to maximize data locality, reducing false sharing, identifying program errors (as a result of unintended true sharing) or accomplishing aggressive privatization.

1 Introduction

OpenMP is a de facto standard for shared memory programming that can be used to program SMPs and distributed shared memory systems. OpenMP follows the wellknown fork-join model, a parallel execution model where teams of threads are created when a program enters a parallel region [16] and terminated upon its completion. Studies [15] have shown that good array privatizations techniques are one of the most effective ways to improve OpenMP performance and scalability. The reason for this is that private data is local to a thread, and systematic privatization will minimize data sharing among threads and thus the cost of cache coherency mechanisms and the long latency incurred when fetching data updated elsewhere in the system. Achieving good privatization in OpenMP might require extensive rearrangement of code, particularly when this is applied to the entire application, as is required by the OpenMP SPMD style [8][9]. Unfortunately this is not an easy task. Prior to an effort of this kind, it is crucial to know how the arrays are being accessed by the executing threads, and determine which regions of arrays are shared between multiple threads. A number of tools have been developed that analyze memory references; however, none of them summarize accesses to OpenMP shared data and display that information to help the user privatize or otherwise study and potentially reorganize memory accesses.

In this paper we present the enhancements made to the Open64 compiler [11] and the Dragon tool [5], a program analysis tool built on top of this compiler, to enable them to collect and represent information on the manner in which threads access the elements of shared arrays at run time. We also describe how we have accomplished this task. In the next section, we briefly describe the current functionality of Dragon and give an overview of Dragon and those analysis modules that we use for the implementation of the tool. Then in Section 3 we focus on the problems of determining and representing array sections in interprocedural code regions. After that we explain how OpenMP is lowered in our version of the Open64 compiler, and how this affects the region calculation and our instrumentation strategy. As part of this section, we provide a simple case study that illustrates our extension to the Dragon tool. Finally, we discuss related work, and present some conclusions and future work.

2 Dragon and Open64

The Dragon analysis tool is a production tool for developing, understanding, and maintaining large scale sequential and parallel programs. It is built on top of our enhanced version of Open64 compiler [11] to displays program analysis information in a intuitive way to application developers who need to understand a given source code in depth. The Open64 compiler was originally developed by Silicon Graphics Inc. and is currently maintained by Intel. It is an optimizing compiler suite for Linux/Intel IA-64 systems. Our version of Open64, Open64.UH, has refined interprocedural analsis module and accepts special flags to extract various program information used in Dragon. The input languages for Dragon are FORTRAN 77/90, C, and OpenMP/MPI. Current supported program analsis information includes static/dynamic call graph, flow graph, data dependence results, automatic performance and feedback instrumentation.

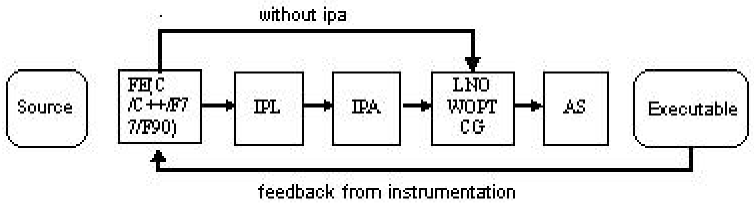


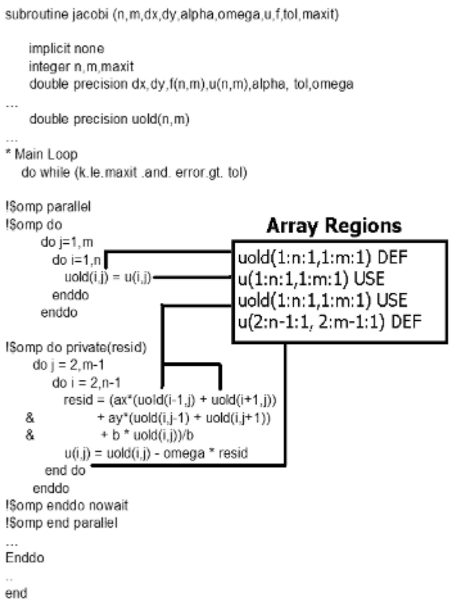
Fig. 1. The modules of Open64 compiler. The user can select interprocedural analysis using the (-ipa flag) or go directly to the backend, where the loop nest and global optimizer reside. The user has the option to automatically instrument a program and obtain feedback files for one or more runs; this information is used for further optimizations

The Open64 intermediate representation, called WHIRL, has five different levels, starting with very high level (VHL) WHIRL, and serves as the common interface among all its components. Each optimization phase is designed to work at a specific level of WHIRL. Our extensions to Dragon use information primarily from the VHL and High Level (HL) WHIRL phases, which preserve high level control flow constructs, such as loops and arrays, as well as OpenMP directives, which are preserved as compiler pragmas. It is very important for our purposes to work in the high level representation because the arrays are still represented explicitly as high level constructs (before they become addresses/pointers), allowing the compiler to perform high level array region analysis calculations. The Open64 compiler consists of five modules as shown in Fig. 1. Multiple frontends (FE) parse C/Fortran programs and translate them into VHL WHIRL. If interprocedural analysis is invoked, then IPL (the local part of interprocedural analysis) first gathers data flow analysis information from each procedure, and then summarizes and saves it in files. Array accesses are summarized into array regions for each individual procedure. Then, the main IPA module generates the call graph and performs interprocedural analysis and transformations based on the call graph data structure. In this phase, the array regions are propagated to determine how the arrays are being accessed across the entire program. If the program is instrumented, the inteprocedural analyzer is able to exploit runtime feedback to optimize the callgraph, in particular to decide when to clone procedures with constant parameters, and for inlining purposes. Our extensions to Open64 were mainly concentrated in the interprocedural analyzer module (IPA), where we retrieve static array regions calculations and in the instrumentation libraries, which we have used to instrument the program at the control flow level using array region information. We will explain this instrumentation process in the following sections and will also show how we have used the analysis within the compiler. The backend module consists of the loop nest optimizer (LNO), the medium-level code optimizer (WOPT), and the code generator (CG). We do not discuss these further here.

3 Array Regions and Interprocedural Analysis

Most of the data processed by scientific programs are stored in arrays. Therefore a good method to represent and manipulate array access information is key to the success of interprocedural data flow analysis. The classical analysis methods [18,19] treat an array as a whole and only use two bits (one for DEF and the other for USE) to represent the accesses to an array. They are very efficient in terms of storage space and computation complexity. However, they are too coarse for many optimizations. Several more accurate methods have been proposed to address this problem. They are largely categorized into exact and summary methods. Exact methods, also called reference-list-based, maintain information about each reference to the elements of an array. Linearization [4] and Atom Images [7] are two implementations of this idea. They are precise, but are very expensive in terms of representation size and operations on them. On the other hand, summary methods require storage space which is independent of the size of the access sets. Well-known strategies for summarizing array regions are Regular Sections [21], Bounded Regular Sections [20] using triplet notations, and Linear-constraint methods such as Data Access Descriptors [22] and Regions [23]. They use geometrical spaces containing accessed array elements to approximate and represent access information rather than storing individual references. Each summary method has its benefits and drawbacks in terms of complexity and accuracy. Flexible shapes usually mean high precision but also complex computation. For example, Triolet's Regions method is fairly accurate because it applies a set of linear constraints to express a convex region. However, the standard operations (union and intersection) to form and compare such regions are very time-consuming.

Open64 adopts Triolet's Regions to perform its array access analysis at both intraprocedural (LNO) and interprocedural levels (IPA). The IPA array regions are summarized and propagated from local region information generated in LNO for global and formal parameter arrays. An IPA level region is created for each type of access mode (USE, DEF, FORMAL (formal parameter) or PASSED(actual parameter)) for each array variable at procedural level, which in turn includes detailed region information at each array access site in form of Projected Regions. A Projected Region consists of a set of triplet notations [LowerBound, UpperBound, Stride](Projected Nodes in Open64) for each dimension. System of linear equations are formed from constraints for each axis of an array using $lower \leq axis \leq upper$. To propagate array region information, there is a mapping between caller and callee. Open64 first performs a reshape analysis and checks if there are aliases and global variables involved. It then propagates information about formal parameters used as symbolic terms in array region summaries, which later will be used to trigger cloning, if possible. A region could be MESSY or UNPROJECTED if it encounters non-linear access or unknown values of bounds.


```
subroutine jacobi (n,m,dx,dy,alpha,omega,u,f,tol,maxit)

    implicit none
    integer n,m,maxit
    double precision dx,dy,f(n,m),u(n,m),alpha, tol,omega
...
    double precision uold(n,m)
...
* Main Loop
   do while (k.le.maxit .and. error.gt. tol)

!$omp parallel
!$omp do
       do j=1,m
         do i=1,n
           uold(i,j) = u(i,j)
         enddo
       enddo

!$omp do private(resid)
      do j = 2,m-1
         do i = 2,n-1
           resid = (ax*(uold(i-1,j) + uold(i+1,j))
    &                + ay*(uold(i,j-1) + uold(i,j+1))
    &                 + b * uold(i,j))/b
           u(i,j) = uold(i,j) - omega * resid
        end do
      enddo
!$omp enddo nowait
!$omp end parallel
...
Enddo
..
end
```

Fig. 2. Open64 calculation of the regions in a Jacobi OpenMP program. Note that there is one region for each access mode per array for a subroutine and it is given in triplet notation

Figure 2 gives an example of the array region analysis in Open64 using a simple Jacobi code. In this subroutine, *uold* is a local array and *u* is a formal parameter array. The DEF region of *uold, (1:n:1,1:m:1)* comes from only one definition site while its USE region *(1:n:1,1:m:1)* is the result of summarizing 6 usage sites. The boundary constraints are used to form system of equations for each region. For example, *u(2:n-1:1,2:m-1:1)DEF* corresponds to two linear equations: $2 \leq dim1 \leq n-1$ and $2 \leq dim2 \leq m-1$.

4 Extensions to Dragon and Open64: OpenMP Early Lowering, Array Regions Calculations and Instrumentation

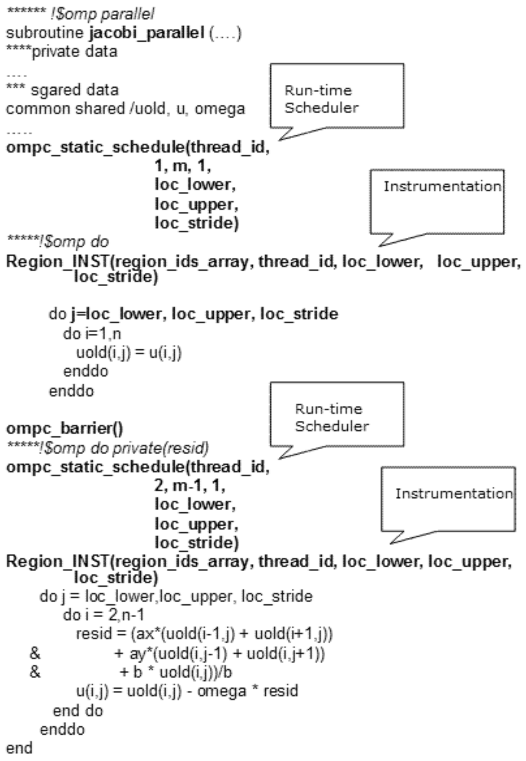
Our extensions to Dragon to provide this new functionality uses the static symbolic analysis and array data flow analysis within the compiler to instrument the program strategically at the control flow level using array region information. Our goal is to calculate the how threads access the array regions in an OpenMP code. To do so requires us to take into consideration how OpenMP is lowered in Open64, since many of the arrays change in scope and definitions (from local variables to global/interprocedural variables, in the case of shared variables). This is the result of the compilation phase where OpenMP directives are transformed into multi-threaded code. It typically involves outlining parallel regions by turning them into procedures containing the code in the region, making shared data available by turning them into global variables or passing them as arguments to the parallel region procedures, redefining private data as local variables to parallel region procedures, and inserting new loop bounds with scheduling calls for parallel loops. [17] discusses a similar strategy for lowering OpenMP in a source to source compiler.

Our extensions to Open64 force the compiler to lower OpenMP directives before the Array region analysis take place. This allow us to accurately calculate the array regions accessed on a thread by thread basis. Since shared data becomes global in Open64's OpenMP lowering process, the array regions accessed are propagated across the procedure boundaries, allowing us to determine how shared arrays are accessed in the entire program. This information helps us to figure out what information we need to gather at runtime and to instantiate any symbolic expressions that describe array Regions. Information such as unknown loop bounds etc. In Figure 3 we show how the Jacobi program looks after OpenMP lowering.

Figure 3 shows how an OpenMP program is lowered. It also illustrates how a parallel region is translated to a procedure that every thread participating in the computation of the parallel region will execute. The run-time schedule assigns temporary upper and lower bounds and a stride to each thread. Since the Jacobi program uses a static scheduling scheme, each thread will get, as far as possible, an even amount of work. This means that the iteration space is divided equally among the threads. Figure 4 shows how the lowering of OpenMP affects the regions of the program in comparison to the original ones in Figure 2. The loop boundaries of the parallel loops are assigned to the threads at run-time.

Now the problem resides in determining the array regions accesses on a per-thread basis. For this purpose, we instrumentthe parallel loops nests that have access to shared arrays. Figure 3 also shows how instrumentation calls are inserted in the lowered version of the Jacobi program. The arguments for the RegionInstrument calls include the thread ID, procedure ID, basic block ID, array region ID, and a list of terms/variables needed, to determine the array region boundaries.

Figure 5 shows the results after running the instrumented Jacobi program using four threads. In this code, you can see which portion of the array *UOLD* is being accessed by each of the four threads. The overlap areas among the regions indicate where the threads share data. This information can help the programmer decide how to partition shared



```
****** !$omp parallel
subroutine jacobi_parallel (....)
****private data
....
*** sgared data
common shared /uold, u, omega
.....
ompc_static_schedule(thread_id,
                     1, m, 1,
                     loc_lower,
                     loc_upper,
                     loc_stride)
*****!$omp do
Region_INST(region_ids_array, thread_id, loc_lower,  loc_upper,
        loc_stride)

     do j=loc_lower, loc_upper, loc_stride
       do i=1,n
         uold(i,j) = u(i,j)
       enddo
     enddo

ompc_barrier()
*****!$omp do private(resid)
ompc_static_schedule(thread_id,
                     2, m-1, 1,
                     loc_lower,
                     loc_upper,
                     loc_stride)
Region_INST(region_ids_array, thread_id, loc_lower, loc_upper,
        loc_stride)
    do j = loc_lower,loc_upper, loc_stride
       do i = 2,n-1
         resid = (ax*(uold(i-1,j) + uold(i+1,j))
   &              + ay*(uold(i,j-1) + uold(i,j+1))
   &               + b * uold(i,j))/b
         u(i,j) = uold(i,j) - omega * resid
      end do
    enddo
end
```

Fig. 3. Open64 lowering of a Jacobi OpenMP program to explicit multithreading code. Note how the parallel region is translated to a procedure, shared variables become global variables, and the loop bounds of the parallel loops are assigned by a run-time scheduler

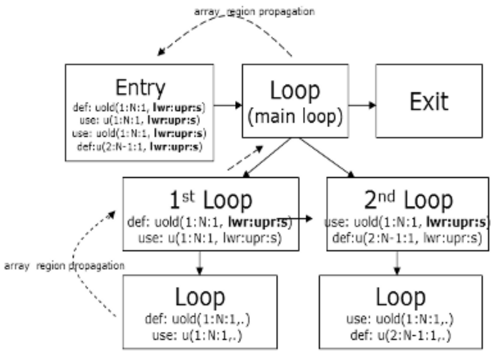


Fig. 4. The array regions of the OpenMP-lowered version of Jacobi. Note the difference between the boundaries of the regions here and the original ones

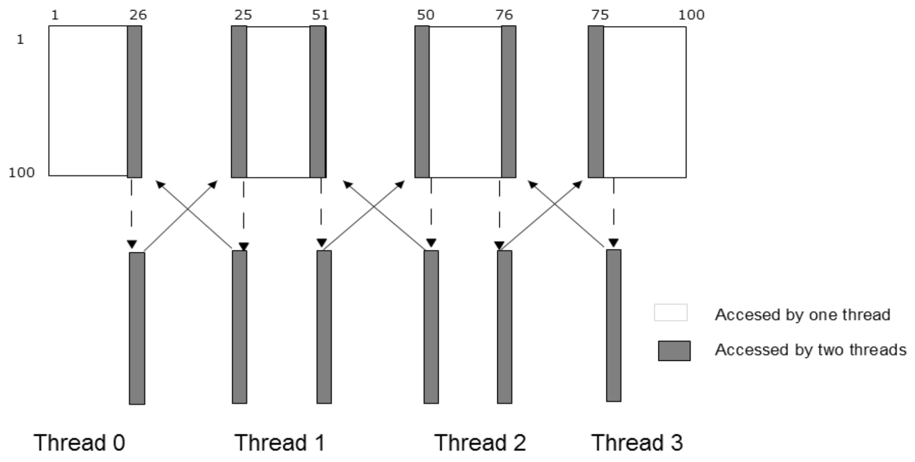


Fig. 5. The different portions of the array *UOLD* accessed by the different threads. The overlap area of the regions is the subregion where the threads share data. The array portions of the regions that are accessed by one thread only can safely be privatized

arrays to maximize privatization. [15], [9] presents an example illustrating how this can be accomplished and describes their experimental results.

5 Related Work

A variety of tools provide detailed information on the execution behavior of a sequential or parallel program, but with different goals and approaches. VTune [6] and PAPI [2] retrieve hardware counter information to poll or take samples to find out how the application is executing, focusing on memory issues such as cache misses. The information retrieved is low level and cannot be used to give a higher-level view of how OpenMP shared arrays are accessed. On the other hand, tools such as MemSpy [10], which map memory accesses to source code and try to understand how to adapt an application to the current memory subsystem, do not summarize array accesses and present them in an intuitive way to the programmer. Intel's ThreadChecker [6] focuses on semantic problems in OpenMP to detect data races, uninitialized private data, and more, but does not emphasize shared data accesses. MetaSim [14] and Dyninst [3] provide a framework to extract the memory signature of programs and try to match the application to the best hardware profile available, but they rely on simulations and explore other problems. Our approach is based on combined static and dynamic analysis and instrumentation to retrieve runtime information for summarizing and displaying the array regions accessed at the thread level.

6 Conclusions and Future Work

Our extensions to the Dragon tool allow us to construct a mapping between OpenMP threads and the regions of arrays that they access. This is accomplished by recalculating the array regions within the compiler after the OpenMP lowering has been performed, and using the region analysis within Open64 to help strategically instrument the application. During execution, the regions can be made precise. There are a variety of potential uses of this information. Our Dragon tool and Open64 compiler version are robust systems that can handle large-scale programs. Further work is needed to evaluate this new feature of the tool on production applications.

References

1. V. Balasundaram and K. Kennedy. A technique for summarizing data access and its use in parallelism enhancing transformations. *Proceedings of the ACM SIGPLAN 1989 Conference on Programming language design and implementation*, 41–53, 1989.
2. S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A Scalable Cross-Platform Infrastructure for Application Performance Tuning Using Hardware Counters. *Proc. Supercomputing 2000* , November 2000, Dallas TX.
3. B. Buck and J.K. Hollingsworth. An API for Runtime Code Patching. *Journal of Supercomputing Applications*, 14(4):317–329, 2000.
4. Michael Burke and Ron Cytron. Interprocedural dependence analysis and parallelization. *Proceedings of the 1986 SIGPLAN symposium on Compiler contruction*, 162–175, 1986.
5. Barbara Chapman, Oscar Hernandez, Lei Huang, Tien-hsiung Weng, Zhenying Liu, Laksono Adhianto, Yi Wen. Dragon: An Open64-Based Interactive Program Analysis Tool for Large Applications. *Proceedings of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT '03)*, 2003.
6. Intel Corporation products for OpenMP. Intel ThreadChecker and VTUNE http://developer.intel.software/products/
7. Zhiyuan Li and Pen-Chung Yew. Efficient interprocedural analysis for program parallelization and restructuring. *Efficient and precise array access analysis*, 24(1):65–109, 2002.
8. Zhenying Liu, Barbara Chapman, Yi Wen, Lei Huang, Tien-hsiung Weng, Oscar Hernandez. Improving the Performance of OpenMP by Array Privatization. *WOMPAT'2002, Workshop on OpenMP Applications and Tools*, 224-259, 2002.
9. Zhenying Liu, Barbara Chapman, Yi Wen, Lei Huang, Tien-hsiung Weng, Oscar Hernandez. Analyses for the Translation of OpenMP Codes into SPMD Style with Array Privatization. *WOMPAT'2003, Workshop on OpenMP Applications and Tools*, 26-41, 2003.
10. M. Martonosi, A. Gupta, T. Anderson. MemSpy: Analyzing Memory System Bottlenecks in Programs. *SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1–12, 1992 Newport, Rhode Island.
11. The Open64 compiler Website http://open64.sourceforge.net
12. William Pugh. A practical algorithm for exact array dependence analysis. *Commun. ACM*, 35(8):102–114, 1992.
13. Randy Allen and Ken Kennedy. *Optimizing Compilers for Modern Architectures, A Dependence-Based approach.* 585–588, Academic Press, 2002
14. A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia and A. Purkayastha. A Framework for Application Performance Modeling and Prediction. *Proceedings of Supercomputing 2002*, November 2002, Baltimore.
15. B. Chapman. F. Bregier, A. Patil and A. Prabhakar. *Achieving Performance under OpenMP on ccNUMA and Software Distributed Shared Memory Systems* Special Issue of Concurrency Practice and Experience, Volume 14, Issue 8-9, September 2001.
16. OpenMP Architecture Review Board. *Official OpenMP Specifications* http://www.openmp.org.

17. M. Soukup. *A source-to-source OpenMP compiler* Master's thesis, University of Toronto, Toronoto, Ontario 2001.
18. J. Barth An interprocedural data flow analysis algorithm *in Conference Record of the Fourth ACM Symposium on the Principles of Programming Languages*,(Los Angeles), Jan. 1977.
19. Keith D. Cooper and Ken Kennedy Efficient computation of flow insensitive interprocedural summary information SIGPLAN Symposium on Compiler Construction 1984: 247-258
20. Paul Havlak and Ken Kennedy An Implementation of Interprocedural Bounded Regular Section Analysis IEEE Transactions on Parallel and Distributed Systems, vol. 2, no. 3, pp. 350-360, 1991.
21. David Callahan and Ken Kennedy Analysis of Interprocedural Side Effects in a Parallel Programming Environment J. Parallel Distrib. Comput. 5(5): 517-550 (1988)
22. Vasanth Balasundaram and Ken Kennedy A Technique for Summarizing Data Access and Its Use in Parallelism Enhancing Transformations PLDI June 1989: 41-53
23. R. Triolet, F. Irigoin and P. Feautrier Direct parallelization of call statements SIGPLAN Symposium on Compiler Construction, July 1986: 176-185

A Model Analysis of a Distributed Monitoring System Using a Multi-formalism Approach

Mauro Iacono[1], Stefano Marrone[1], Nicola Mazzocca[1], Francesco Moscato[1], and Valeria Vittorini[2]

[1] Seconda Università di Napoli, Dipartimento di Ingegneria dell'Informazione
Aversa (CE), Italy
{mauro.iacono,stefano.marrone, nicola.mazzocca,francesco.moscato}@unina2.it

[2] Università di Napoli Federico II
Dipartimento di Informatica e sistemistica, Napoli, Italy
vittorin@unina.it

Abstract. Automated applications for environmental monitoring are an important aid for security and safety of buildings and for civil (domotic) and industrial use. The relevance of such applications is sound, and ensuring a correct fulfilling of the mission goals implies the need for performability and dependability requirements. For reasons of cost, flexibility and efficiency, a good candidate architecture is based on mobile agents capable of real time operation on a network of intelligent sensors, in addition of normal monitoring tasks.
Design and tuning of these systems is a very hard task. In this paper a multi-formalism approach based on model-checking techniques for Timed Automata and Timed Petri Nets is presented to model a real-time-mobile-agents based monitoring system. With this approach, different component of the system may be modeled by using the most suitable modeling formalism to cope with the need for modeling agent behaviors, real-time constraints and the load of the overall system.

1 Introduction

Monitoring Systems (MS) are in charge of surveying conditions of industrial environments as well as civil buildings. They exploit a network of sensors, connected by different network architectures and protocols and controlled by heterogeneous computing nodes, with event-driven reactions to gathered data in order to detect an manage with dangerous conditions. The scope of these systems moved from industrial field to domotic applications broadening the market and the quest for inexpensive and flexible systems.

The sensor network is usually devoted to service other periodical tasks in MS, and the reaction is obtained by proper supplemental real time tasks responsible of raising alarms, operating reconfigurations or acting on the monitored process. The nature of the surveillance requirements usually results in the presence of different kind of sensors (optical, thermal, pressure sensors etc.), of different kind of computing nodes (PC, micro-controllers, PLC etc..), of different operating systems (OS) (embedded OS, real-time OS, monolithic and micro-kernel OS), making the whole system very heterogeneous. In addition, usually for physical and security reasons, the system is distributed through the space and different kind of network (deterministic as well as stochastic) links (Ethernet, token ring, proprietary bus for sensors, serial lines etc.) connect nodes, sensors and actuators in the system.

The need for flexibility, reconfigurability and adaptability can be better satisfied by approaching the design by the *mobile agents* [9,10] programming paradigm, which offers a way to put the focus on the solving strategy more than on the distribution of the architecture and is supported by a sound theoretical support for modeling system behaviors. A mobile agents oriented approach must be conciliated with the need for real time specifications satisfaction, then an accurate quantitative design for system performances has to be performed, as well as formal technique must be used to ensure that the system always works correctly and in a safe state.

Design and tuning of these systems is a very hard task since network and OS scheduling behaviors have to be considered in addition to monitoring tasks and other concurrent system tasks.

To better cope with the complexity of such systems, multi-formalism approaches are emerging, which allow to use different formalisms to better model and analyze different parts of a system and also to promote model reuse.

In this work we propose a methodology for real time, mobile agent oriented, heterogeneous systems performance design based on a multi-formalism approach. We based our methodology on Timed Automata (TA) [1,2] and Timed Petri nets (TPN) [5] in order to model system behavior against hard real-time constraints, accounting for operating systems scheduling effects, network communication queuing effects and concurrency among tasks.

Timed Automata became a standard in modeling real-time asynchronous systems [1,2,7] but they do not offer features to model queuing effects, as needed to obtain a full evaluation of system behavior. It is mandatory to fill this gap, because scheduling affects system performances, and to find out a formal means for the evaluation of *clock* values (see [1,2]) needed to describe the time elapsed when the automaton is in a state. The evaluation of this values also has to consider concurrency among tasks. Formal techniques based on Petri Nets (PN) are widely used to model concurrent and distributed systems, and showed to be effective in many research papers as [4,3] in which they are used to model FSM systems.

The aim of our methodology is to describe monitoring system by means of TA where clocks values are determined by developing appropriate TPN, in order to limit the explosion of computing time and to limit the increase in complexity for the model. To obtain this, we structure in the analysis and design process the whole model in layers, to each of which is associated the formalism that better suits to manage the layer characteristics.

In the next sections we describe the general system architecture, present our modeling approach, together with a laboratory case study.

2 System Architecture

The general architecture to which we refer for these systems may be sketched as in Fig. 1. The architecture is composed by computing units that can be divided in two groups: sys-

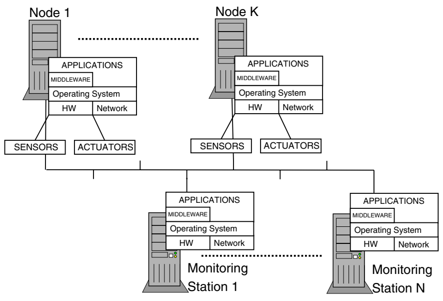


Fig. 1. Monitoring Systems Architecture

tem nodes and monitoring stations. With reference to Fig. 1, $Node_i, i \in 1..k$ are computing nodes of the system which host sensors and actuators, while $MonitoringStations_i, i \in 1..N$ are computing nodes on which supervisory user interfaces and control panels run. The system to be monitored and the monitoring stations are usually distributed and heterogeneous. Different hardware(HW), network links and protocols (*Networks*), OS (*Operating Systems*) and software may be used in these systems. System nodes may be embedded systems (microcontrollers, PLC etc.) as well as general purpose Personal Computers (PCs). Since the monitoring system architecture is fully distributed, all nodes are involved in control, supervisory and monitoring tasks.

Applications running on the nodes can use distributed *middleware* for communications, to share or virtualize some node resources.

In addition, applications have to be executed under (soft and/or hard) real-time constraints. Node Operating systems and middlewares have to provide means for the application to be compliant with such constraints in a distributed environment where tasks are not scheduled in a fixed and periodical way. Moreover, monitoring system should provide adaptability and reliability features, in order to face also with anomalous system behaviors and to supply the best control and monitoring effort depending on the current system state.

In the architecture proposed in this paper the principal middleware is a mobile agent framework, chosen to improve the reliability and the adaptability of static monitoring applications. Both static agents and mobile agents are active in the system. Static agents are responsible of standard monitoring and control operations. Mobile agents are used to scan all sensors (optical, thermal etc.) in the distributed environment. While scanning all system sensors, mobile agents may reveal unusual system behaviors and can try to reconfigure local control systems to deal with them. Interactions among agents are performed if some agents need the intervention of other agents features to accomplish their tasks. Tasks performed by agents usually have real time constraints; furthermore agents have to compete for resources with standard monitoring and control applications. Real time constraints and priorities for all standard task and for all agents have to be carefully defined to avoid non correct behaviors.

3 Modeling Approach

The main goal of our approach is to define a methodology to support the modeling and analysis phases of complex real-time, heterogeneous and distributed monitoring systems shown in the previous section. The methodology we present is founded on formal methods, and is based on a modular multi-formalism approach, in order to exploit both performance modeling and model checking techniques advantages together. We chose a modular approach in order to help the designer to analyze the system to be modeled by submodels, in a divide et impera strategy and enabling submodel reuse.

The main problem in modeling this kind of system is to prove in early design phase that real-time constraints are met in the whole system, considering all different (potential) component interactions at different level, for example at operating system scheduling level, at middleware level or at application level. In a simple scenario, a stand alone monitoring software (a static agent) and a mobile agent used to avoid extraordinary dangerous situations can be scheduled and then executed concurrently on the same node. If the stand-alone application and the mobile agent use the same sensor, a preemption mechanism must be designed in order to obtain in turn exclusive access to the resource. In addition it is necessary to prove that the real time constraints stated in the specifications are still met by each agent also when other application agents operate on the nodes and interact with each other by communicating among nodes.

Moreover, to guarantee a correct behavior of the whole system, it is not sufficient to guarantee that each single component mets the specified constraints but also that the various components still met the constraints when they execute in the distributed environment: scheduling, interactions and concurrent executions obviously modify the response time of each component on each node; in addition, undesired deadlocks and other incorrect behaviors that can be introduced by the interactions among different applications and drivers must be avoided. The steps of the proposed methodology are sketched in Fig.2.

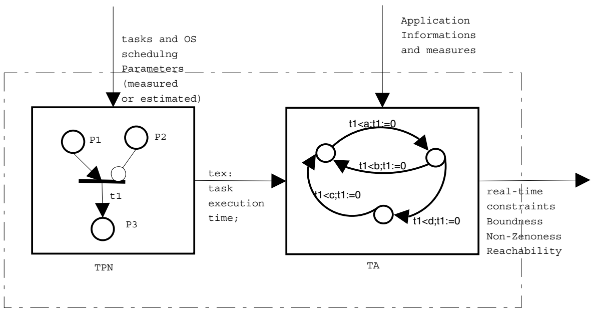

tasks and OS schedulng Parameters (measured or estimated)
Application Informations and measures
P1
P2
t1
P3
TPN
tex: task execution time;
t1<a;t1:=0
t1<b;t1:=0
t1<c;t1:=0
t1<d;t1:=0
TA
real-time constraints Boundness Non-Zenoness Reachability

Fig. 2. Modeling Approach

- The first step is devoted to evaluate the task execution time of each agent when interacting with other agents on a single node. This characterization is achieved by TPN models, tuned with measured execution times for elementary actions performed by the agent. In addition, OS scheduling effects are accounted in the model. TPN are used in order to model queuing effects and priority scheduling aspects in the overall node behavior. Even if TPN formalisms is capable of representing the whole system behavior, interactions among nodes are non modeled by this formalism because of the state space explosion problem which heavily affects solution of PN models ([8]).

- The second step models interactions among agents acting all over the system by TA. This choice is a way to reduce the effects of the state space explosion using optimized algorithms and data structures ([7]). Model checking techniques are applied to verify properties on the whole system behavior. The properties to be evaluated for the whole system are expressed in the Real-Time Computational Tree Logic (RTCTL). In this way properties such as *Reachability*, *Non-Zenoness*, *Bounded Response* ([6]) in addition to real-time constraints compliance can be evaluated on the whole system.

The approach needs to be improved by using the first step of this methodology because it is very difficult to model operating system queuing effects on the tasks that are executed on the same node by TA. Results of the first step (task execution times on single nodes) are used to set guard transition times of TA models, together with measured communication and migration times for agents. The final result is a validation of the system against specifications in terms of satisfaction of the aforementioned properties.

4 Case Study

We applied the methodology described before to a security and safety control system. The system is used to detect physical intrusions (revealed by pressure sensors) and firing emergency in a building. On system nodes there are three different types of sensors: pressure, thermal and optical (camera). These sensors are controlled by embedded systems (that are emulated in the laboratory test-bed by low-performance PCs: Intel Pentium processor at 166 MHz, 32 Mbyte of RAM and 100 Mbit Ethernet). These computers are equipped with Slackware Linux based on kernel 2.4.22.

Five kinds of software agents are active in our test-bed. The first kind and the second are constituted by static agents which respectively monitor thermal and pressure sensors connected to the node on which they operate. The third kind is composed by static agents which continuously move optical sensors in a predefined space. The fourth kind is constituted by mobile agents which wander on the system monitoring all thermal sensors looking for and reacting to dangerous situations, while the fifth and last kind groups mobile agents which patrol the system looking for pressure anomalies to locate intruders.

In Fig.3 an alarm condition to analyze is shown: the mobile agent monitoring the thermal sensors (*Nomad*) detects an alarm condition after preempted the static agent (*ThermSW*) that was monitoring the same sensor. In order to assure that no human is present in the room before locking doors, *Nomad* requests the help of the mobile agent that monitors pressure sensors (*Nimrod*). *Nimrod* moves to the *Nomad* node and preempts the static pressure agent that was monitoring the pressure sensor (*PressSW*). Then *Nimrod* sends a message to *Nomad* notifying if humans are present in the room or not. Depending on *Nimrod* message, *Nomad* decides if it is possible to lock the anti-fire door or not.

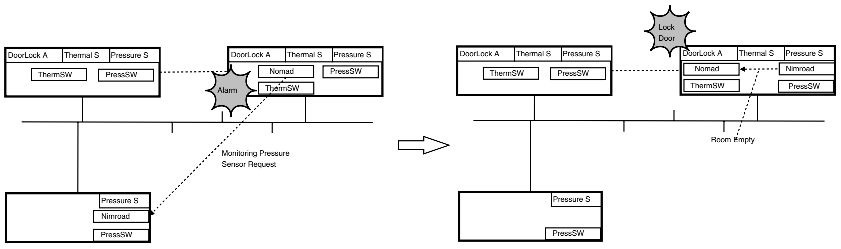


Fig. 3. Case Study Example

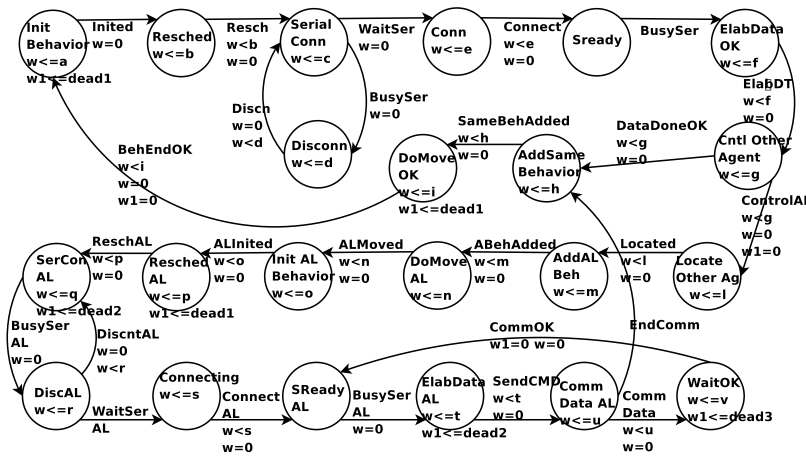


Fig. 4. Nimrod TA

It is necessary at design time to evaluate if this operation can be executed within in 10 secs. In addition it is necessary to prevent deadlocks and livelocks in the whole system. For these reasons a TA model of each system component behavior is realized and all the (sub)models are composed to build the whole system model. In Fig.4 and Fig.5 parts of the Nimrod and Nomad TA are respectively shown (the other models are not reported for brevity sake). In particular Fig.4 shows the part of *Nimrod* TA that models its behavior when no intruders are detected (states from *init* to *ElabDataOK* models the interaction with the sensor serial interface). *Nimrod* continues its normal behavior if *Nomad* does not require its help (when from the state *ControlOtherAgent* follows the transition *Data-DoneOK*) or moves to *Nomad* node if it needs to know if some humans are in its room after a fire alarm detection (when from the state *ControlOtherAgent* follows the transition *DataDoneAL*). If no help is required, *Nimroad* continues with its normal behavior (*AddSameBahavior*) moving (*DoMoveOK*) to the next node. If *Nomad* requests its help, *Nimrod* changes its behavior to accomplish its new task (*AddAlBehavior*) and moves to the *Nomad* node (*DoMoveAl*) to monitor pressure sensors, also preempting Pressure static agent on the *Nomad* node (from *ReschedAl* to *SReadyAl*). It elaborates pressure data and communicates the needed informations to *Nomad* (from *ElabDataAl* to *WaitOk*).

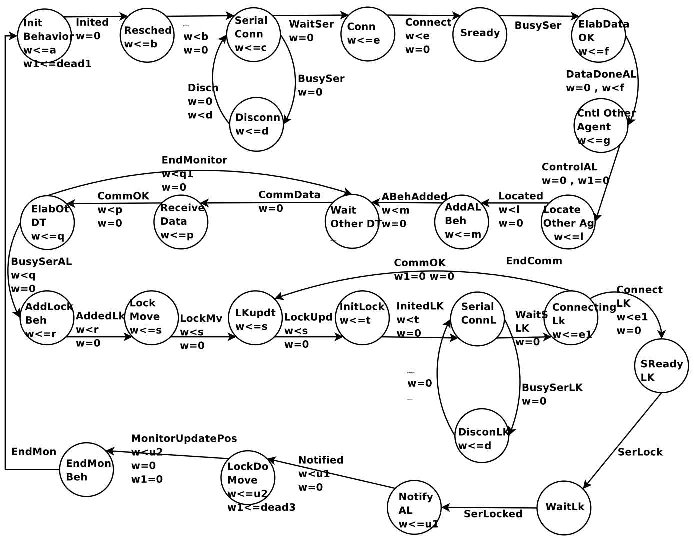


Fig. 5. Nomad TA when an alarm occurs

In the same way Fig.5 shows the parts of Nomad TA that models its behavior when a fire alarm is detected. Differently from *Nimrod*, when *Nomad* finds an alarm condition, it changes immediately its behavior (*AddBehAl*), requesting *Nimrod* intervention (from *WaitOther* to *ElabOtherData*). In Fig.5 the part of *Nomad* TA concerning its behavior when no human is detected by *Nimrod* is shown. It moves to the node interfaced with the door locking mechanism and it locks the door (from *AddLockBeh* to *LockDoMove*). Different real-time constraints to verify are reported in the two timed automata; in particular, the state invariant $w_1 <= DEAD_3$ in the State *LockDoMove* in Fig.5 represents the design constraints that states that a decision for a lock of a door in the case of a fire-alarm must be taken within 10 secs (where $DEAD_3 = 10secs$). Some interesting properties to prove on these models (once composed in the whole system model with other TA submodels) are *Reachability* of a given set of states and *Bounded response* to a request starting from a given state. These properties are expressed in RTCTL and the tool KRONOS [6] is used to check the properties on the model. For example, it is useful to assure that once *Nomad* has retrieved an alarm condition, no deadlocks or livelocks or other dangerous conditions prevent *Nomad* to lock the door. This can be expressed in RTCTL:

$$initBehavior \Rightarrow \forall\triangle EndMonBeh$$

where "$\Rightarrow \forall\triangle$" means: from *InitBehavior* state *every state along every execution* brings to *EndMonBeh*state . Another properties to prove is that once *Nomad* has detected a fire-alarm, in a time less than or equal to 10 seconds It have to Lock the anti-fire doors. This can be expressed into RTCTL in the following way:

$$ElabDataAl \Rightarrow \forall\Diamond_{\leq 10} EndMonBeh$$

Table 1. Some Experimental Results

Parameter	*Max val measured on real system*	*Deadline*
$DEAD_1$	2.17 secs	3 secs
$DEAD_2$	4.12 secs	5 secs
$DEAD_3$	8.1 secs	10 secs

where "$\Rightarrow \forall\Diamond_{\leq 10}$" means: from *ElabDataAl* state *same state along every execution* brings to the state *EndMonBeh* within 10 seconds.

In order to prove these properties, the whole TA model have to be model checked and to accomplish this task it is necessary to : (1) tune the TA submodels; (2) compose the TA submodels to build the whole system TA; (3) model check the system to evaluate the needed properties. In order to accomplish the task at point (1):

- design-time (real-time) constraints are used to tune state invariant values (for example the value of $DEAD_3$ in Fig.5) - Values retrieved from measures on prototypical implementation of part of software agent are used to tune some transition guard time value (like as the value of variable i in Fig.4 that represents the time that an agent spends to move from a node to another one) - TPN models are used to estimate the values that depends on operating system scheduling policies and mechanism and on the presence of other software agents on the same elaborating node (like t or p variables in Fig.4). Fig.6 shows two parts of the TPN model for the monitoring system: on the left a TPN submodel of *Nimrod* mobile agent behavior is shown, while on the left a TPN submodel of operating system scheduler is shown; the other submodels are omitted due to the lack of space. These submodels are composed to estimate response time of agents when they are scheduled on the same elaborating node, Throughput is same transition are then used to estimate some values used to tune the TA model (for example, the inverse of Nimrod TPN $t_{readSensor}$ throughput is used as value for variable t of the *Nimrod* TA model of Fig.4).

Points (2) and (3) are performed by submitting the (tuned)TA submodels directly to the KRONOS toolkit.

Results from Kronos toolkit assure that all requested properties are satisfied. In particular, the results about bounded executions (concerning deadlines of 3 and 5 seconds respectively for $DEAD_1$ and $DEAD_2$ in Fig.4 and a deadline of 10 seconds for $DEAD_3$ in Fig.5) assures that the deadlines ever met. In addition a prototypical system was built as described at the beginning of this section and experimental results show that effectively the system verify the needed properties : no deadlocks or livelocks are detected in the system and all real-time constraints are met. Some experimental results concerning the deadlines described previously are shown in Tab.1.

5 Conclusions

The presented modeling methodology, applied to the case study of a remote SCADA system, efficiently and effectively allows to define and use multi-formalism models by

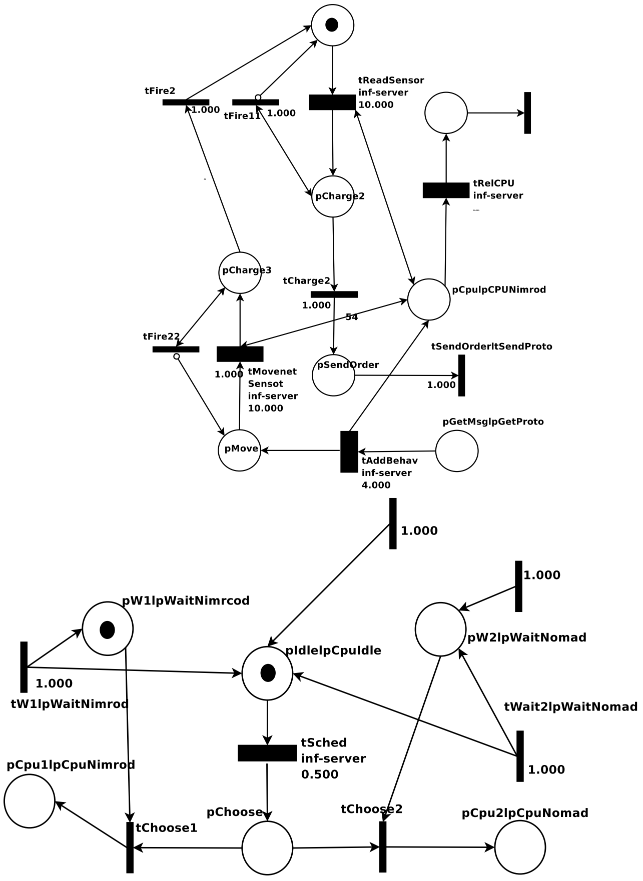


Fig. 6. Nimrod and Priority Scheduler TPN

composing them from heterogeneous submodels to build a model of a complex supervisory system. The proposed methodology allows to use different formalism for different problems in different levels of modeling and provides a modular approach to system analysis, supporting reuse of submodels. As shown in the *Case Study* section, (sub)models are simple and are solved independently with advantages for the evaluation phase. Preliminary results performed on a reduced system architecture showed the effectiveness of the technique on the partial implementation. A proper model checker has been designed and implemented in order to allow the verification of desired properties against the system and further laboratory experiments are being pursued on the whole architecture. Future works include the integration of the technique in a multi-formalism modeling framework in order to automate the solution process and to evaluate the opportunity of introduction of other kind of models (including also simulative models) in the methodology itself and to define all the composition rules to perform their integration.

References

1. Alur R., Courcoubetis C. and Dill D.L. "Model Checking for real-time systems". In *Proc. of the 5th Annual Symposium on Logic in Computer Science*. IEEE Computer Society Press, pp. 414-425, 1990.
2. Dill D.L. "Timing Assumption and verification of finite-state concurrent systems" In *Proc. of the International Workshop on Automatic Verification Methods for Finite State Systems*. LNCS 407, pp. 197-212. Springer, 1989.
3. Wang J., Deng Y. "Incremental modeling and verification of flexible manufacturing systems". In *Journal of Intelligent Manufacturing*, Vol. 10, No. 6, pages 485-502. Dec. 1999.
4. Lin M.-H. and Fu L.-C. 1999. "Modeling of Priority Queuing Service in discrete Event System Using Hybrid Petri Nets". In *Proc. IEEE International Conference on Systems, Man and Cybernetics* Vol.1.
5. Raymond M. and Alain J.-M. 1993: "Quantitative Evaluation of Discrete Event Systems: Models, Performances and Technique"s. In *Proc. 5th International Workshop on Petri Nets and Performance Models* (19-22 Oct 1993), IEEE, 2-11.
6. Daws C., Oliviero A., Tripakis S. and Yovine S. "The tool KRONOS". In *Hybrid Systems III: Verification and Control*, LNCS 1066, pp.208-219. Springer, 1996.
7. Clarke E.M., Grumberg O. and Peled D.A. "Model Checking" , The MIT Press, 1999, Cambridge
8. Murata T. "Petri Nets: Properties, analysis and applications" In *Proceedings of IEEE*, 77(4).
9. http://jade.cselt.it/
10. http://www.trl.ibm.co.jp/aglets

Performance Oriented Development and Tuning of GRID Applications

Emilio Mancini[1], Massimiliano Rak[2], Roberto Torella[2], and Umberto Villano[1]

[1] Universitá del Sannio, Facoltá di Ingegneria
C.so Garibaldi 107, 82100 Benevento, Italy
{epmancini,villano}@unisannio.it

[2] DII, Seconda Universitá di Napoli
via Roma 29, 81031 Aversa(CE), Italy
{massimiliano.rak,r.torella}@unina2.it

Abstract. GRID Application development is a hard task. Good applications should correctly use large distributed systems, whose infrastructure heavily affects the application performance. In this paper we propose a performance oriented approach to GRID application development, founded on the use of a prototype language (MetaPL) for the description of the applications and the use of a heterogeneous system simulation environment (HeSSE) for performance prediction. We developed GRID simulation components for the existing simulation environment (HeSSE) and validated them. After that we extended the MetaPL language in order to explicitly support GRID application features and simulated a simple case study to show how the approach works.

1 Introduction

The presence of distributed software systems is pervasive in current computing applications. In commercial and business environments, the majority of time-critical applications has moved from mainframe platforms to distributed systems. In academic and research fields, the advances in high-speed networks and improved microprocessor performance have made clusters or networks of workstations and Computational GRIDS an appealing vehicle for cost-effective parallel computing. However, the systematic use of distributed programming can be frustrating, especially if the final application performance is more than an issue. Even if great effort has been putting in developing methodologies and tools that could help the final programmer to develop application independently from the underlying architecture, as happen in GRID environment, very few results have been obtained to support prediction and evaluation of prototypal application. In the last few years, our research group has been active in the performance analysis and prediction field, developing HeSSE [6-7], a simulator of distributed applications executed in heterogeneous systems, and MetaPL a prototypal-base language, based on XML, able to support many different programming paradigm. This paper presents a simulation-based methodology, founded on HeSSE and MetaPL, that makes it possible to predict GRID application and system performance, even when the execution environment is not available and the application is not completely developed. This methodology can be used as the basis for performance-driven GRID application development or GRID system performance tuning and design. The reminder of the paper is structured as follow: next section will show briefly the related work on the GRID simulation. Section 3.2 will describe our simulation environment and modeling technique and the newly developed the GRID extensions; the section will point out the simulation components developed and their validation. Finally the approach will be applied on a simple case study and then compared to the actual results obtained in the real (i.e., non-simulated) GRID environment, discussing the accuracy of the model used and the effectiveness of the proposed approach. The paper ends with a section on conclusions and future work.

2 Related Work

The performance analysis and tuning of applications, along with the optimization of scheduling and resource allocation algorithms, are widely recognized as particularly hard problems in Grids. Long application running times, non-repeatability of tests, not to mention economic problems, prevent the use of real applications running on real hardware to grade the effectiveness of alternative solutions. Most of the contributions in literature try to predict application running times, network load and end-to-end data transfer times by statistic models of historical data. An alternative, and probably more manageable solution, is to resort to simulation environments that enable reproducible, controlled and systematic evaluation of middleware, applications, and network services for the Grid. However, none of the simulations environments currently available seems able to provide accurate, fast and robust performance evaluations and analysis of full-scale Grid applications and middleware. In particular, the objective of the Bricks project [4] is to simulate alternative scheduling policies for client-server systems that provide remote access to computing services over the Grid. Bricks makes it possible to simulate alternative resource allocation strategies and policies for multiple clients, multiple servers scenarios. However, Bricks follows centralized global scheduling methodology and hence it is not suitable for simulation of environments where there are multiple un-coordinated schedulers. Microgrid [5] is a simulation environment that provides a virtual grid infrastructure for the study of Grid resource management issues. In fact, it is actually an emulator, in that actual application code is executed on a virtual Globus environment. While on the one hand this characteristic leads to high accuracy results, on the other it affects negatively simulation speed. In practice, most applications are executed in Microgrid in a time longer than the one required in the actual environment, thus making the use of the simulator not viable to perform large number of experiments. Simgrid [6] is instead a C language-based toolkit for the simulation of application scheduling. It supports the modeling of time-shared resources, taking into account the load which can be described synthetically or obtained by previously-collected real traces. As Bricks, Simgrid makes it possible to model environments where there is a single scheduling entity, with the further constraint that the systems must be time-shared. Hence it is not directly utilizable for simulating multiple competing users, applications, and schedulers with different policies, as in most Grid environments, not to mention space-shared machines. Gridsim [3] is the most recent proposal, and it extends and enhances the previous systems, providing modeling of heterogeneous time- and space-shared resources, multiple static or dynamic schedulers, definition of CPU processing power. However, it is fairly limited as far as network and I/O devices simulation is concerned.

3 GRID Simulation with MetaPl and HeSSE

HeSSE is a simulation tool that, using a compositional modeling paradigm, allows the user to simulate the performance behavior of a wide range of distributed systems for a given application, under different computing and network load condition.

The compositional modeling approach allows to easily describe Distributed Heterogeneous Systems that are modeled by interconnecting simple components. Each component reproduces the performance behavior of a section of the complete system at a given level of detail. A HeSSE component is basically an object, hard-coded with the performance behavior of a section of the whole system. More detailed, each component has to reproduce both the functional and temporal behavior of the subsystem it represents.

In HeSSE, the functional behavior of a component is the service set that it exports to the other components. So connected components can ask other components for services. The temporal behavior of a component describes the time spent servicing. System modeling is performed primarily at the logical architecture level. For example, physical-level performance, such as the one resulting from a given processor architecture, is generally modeled with simple analytical models or by integral, and not punctual behavioral simulation. In other words, the use of a processor to execute instructions is modeled as the total time spent in the processor without considering the per-instruction behavior. Thanks to the chosen approach, HeSSE is capable of describing easily very complex Distributed Heterogeneous Systems at any given level of detail.

HeSSE uses traces to describe applications. A trace is a file that records all the actions of a program relevant for simulation. For example, the trace for an MPI application is a sequence of CPU burst and requests to the run-time environment. Each trace is the representation of a specific execution of the parallel program.

Trace files are simulation-oriented application descriptions, usually obtained through application instrumentation. When the application is in development state, they can be generated using prototypal languages. In the past years we developed an XML-based language for parallel programs description: MetaPL [10]. It is language independent and can easily support many different programming paradigms or communication libraries. It is possible to extend the language, through *Language Extensions* XML DTDS, which introduce new constructs to the language; available examples are PVM, MPI and OpenMP language extensions. Moreover MetaPL is able to generate HeSSE trace files through a filter mechanism.

Detailed description of the MetaPL approach to program description and trace generation is out of this paper scope and can be found in [13,12].

The application analysis process can be represented graphically as in Fig. 1. It is subdivided in three steps: System Description, Simulation and Results Analysis. Analysis can drive to new models and process analysis repetition.

The System Description phase includes:

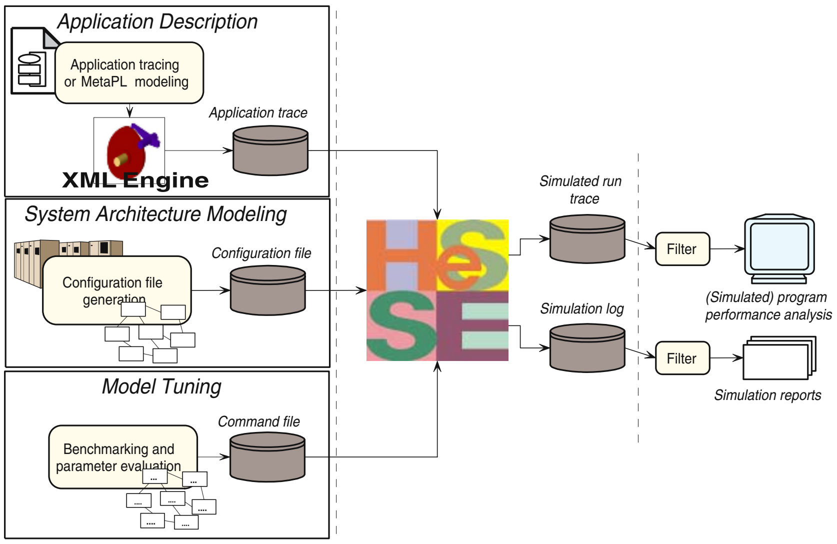

Application Description
Application tracing or MetaPL modeling
Application trace
XML Engine
System Architecture Modeling
Configuration file
Configuration file generation
Model Tuning
Command file
Benchmarking and parameter evaluation
HeSSE
Simulated run trace
Simulation log
Filter
Filter
(Simulated) program performance analysis
Simulation reports

Fig. 1. HeSSE Simulation Session

- MetaPL prototypes development (Application Description);
- system architecture model definition (System Architecture Modelling);
- evaluation of the time parameters (model tuning).

The application description step consists in the MetaPL prototype development. Thanks to the XML prototype, it is possible to generate the trace files needed to drive the simulation execution. Second point consists in choosing (developing, if needed) HeSSE components useful for the simulation, and in composing them through a configuration file; at the end of this step, we are able to reproduce the system evolution. Last step consists in running benchmarks on the target system in order to fill the simulator and the MetaPL description with time information.

We built a simple MetaPL extension that supports GRID specific information, similar to the data that can be retrieved from a globus RSL file, like the number of gatekeeper involved. GRID MetaPL commands will be used by the trace generation mechanism to define high level behavior, like the number of parallel tasks started in the GRID environment.

In order to simulate the GRID infrastructure (i.e. the components of a real GRID environment) we developed a new set of HeSSE components. Following sections will give details about the simulation models adopted.

3.1 GRID Components in HeSSE

Our aims in this paper is to predict the performance of a given application (not completely developed) in a GRID environment in terms of the overall application response time, the time effectively spent in execution or the time needed to be started on the GRID environment.

Basic components in HeSSE (see [8,9]) are able to reproduce main features of common cluster systems, like ethernet and myrinet networks, operating system scheduling and job management, process synchronization and message passing software layers. To reproduce the full GRID environment infrastructure overhead, we need to choice a real implementation to mimic; in the following we will focus on the globus GRID platform solution, mainly on two components: `GRAM` and GIS. The first one, `GRAM`, manages the application allocation on the target system. The `GIS`, component manages the GRID environment security problems, mainly the user authentication. Good description of the cited components can be found in [1].

We developed three libraries containing a number of components needed to a complete simulation. These components are:

– *GlobusClient*: formally the user submitting single or batch jobs to the system.
– *GlobusServer*: this component simulate the gatekeeper in the globus environment.
– *GlobusProxy*: this one simulate the ability of the gatekeeper to connect to other gatekeepers in the case that more, or different, clusters need to be used to complete the job.
– *GPAM (GRID Process Allocation Manager)*: the work carried out by this component is to allocate processes on a cluster, execute them or waiting for an external synchronization if more clusters are used. It formally simulate the GRAM section in the globus environment.
– *Barrier*: it simulate the synchronization infrastructure, `DUROC` in globus, needed to guarantee the contemporaneous execution of all the processes in a multi-cluster application.
– *SSL*: obviously simulate the overhead due to secure communication in a GRID environment.

Now let's have a deeper look to how the simulation is carried out. When the simulation starts, the GlobusClient sends a job execution request to his default GlobusServer and waits for the results. The communication between the client and the server is secure so the SSL component is used. When a server receive a request, verify if he has a GPAM and if it is free or busy. From the request, the server sees if the job must be executed on a single cluster or on multiple clusters. If the server's GPAM is free and the job must be executed on a single cluster, the server sends the request to his GPAM that allocate the tasks on his cluster's machines. If the server GPAM is busy, the server sends the request to his GlobusProxy asking for forwarding to another server. In the case that the job must be executed on multiple clusters, the server sends the job to his GPAM. The GPAM allocates the tasks and waits for the Barrier synchronization. Then the server sends the job to proxy which forward it to the other servers. When all the tasks have been allocated, the Barrier unlock the GPAMs that start the task execution. At the completion, each GPAM sends to his server the results that are forwarded back to the first server and then to the client.

The autenticatio process heavily affects the system performances, iIn order to show the kind of interactions that take place in the startup phase of the application, figure 2 exploits the message exchange in the authentication process, when two different GRID environments are involved. Simulation environment reproduces the complete messag exchange. A detailed verifying process was carried on, monitoring both real environment and the simulation to grant that exactly the same messages are sent at any layer of the network stack.

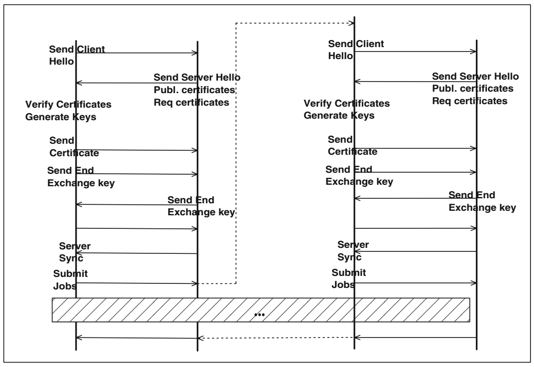


Fig. 2. Message exchange in globus autentication

3.2 Simulation Validation

To validate the correctness of the infrastructure simulation, we tried a simple test application, the unix command *True*, a standard GNU application which simply terminate with error level zero. In that way, the application execution time is almost nought and the measured response time is all due to processes allocation through the clusters. We launched the test application on the target environment (an SMP cluster better described later) in many different configurations. One of the configurations was used to tune the simulation parameters, the others were used to verify the simulation.

To emulate many GRID environment on a single cluster, we used each node as a different GRID cluster. Varying the number of tasks allocated on the GRID allows to show peculiarity in the allocation time. In particular, has been noted that, from two to eight tasks, the allocation time vary linearly while there is a big gap between one and two. This is due to the fact that all the job submission to multiple clusters are made in parallel by the first cluster.

Figure 3 shows both the real environment and the simulation behavior, varying the number of GRID nodes. Upper side of the figure shows the response time, while lower side contains the percentage simulation error. Note that the behavior showed by simulation correspond to the real system one.

4 An Example: The Gauss-Siedel Application

Aiming at clarify how proposed development approach works, we have applied it on a single case study: development of the Gauss-Siedel method for resolving iteratively linear equations systems. We modeled and simulated the target application. After the analysis, we developed and ran it on the real environment. The execution results were

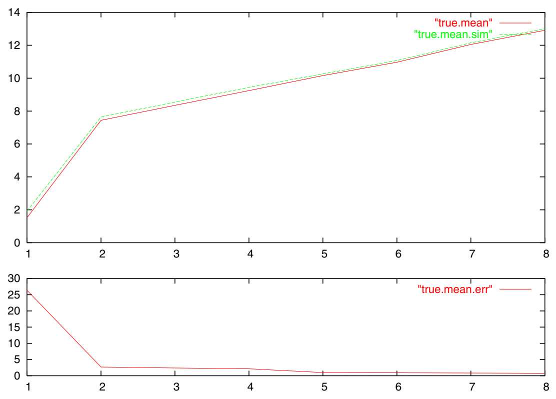


Fig. 3. Gauss-Siedel Simulation

compared with the predicted ones. As previously pointed out the development methodology founds on the following steps:

Application Modeling MetaPL Description of the target application
GRID Environment Modeling Definition of one (or more) HeSSE configuration able to reproduce the target environment(s)
Model Tuning Execution of simple application benchmark able to define the time parameters for the developed models (see [14] for further details)
Application Analysis Application Simulation on the target(s) GRID environments, in order to understand final application performance behavior.

The Gauss-Siedel method for resolving iteratively linear equations systems calculate, at each step, the new unknowns values using those calculated at the previous step. At first step a set of random values is chosen. It works under the assumption that the coefficient matrix is a dominating-diagonal one. Due to the particular method chosen, the parallelization is very simple. In fact each task calculates only a subset of all the unknowns and then gathers with the other tasks the rest of the unknowns vector needed for the next step. Figure 4 shows a partial MetaPL description of the above described application (in order to improve readability, we cut away XML tags and code section not useful for code understanding).

In order to validate the approach, the proposed application was developed and run on a real (even if little) GRID environment: the cluster Cygnus. This cluster is composed of four Pentium III bi-processor nodes with 512MB of memory each and a front-end Pentium IV Xeon with 1GB of main memory. The nodes are connected each other through a switched 100Mbps ethernet LAN while the frontend has two ethernet cards, one private, connected to the switch, and the other public connected to the external world.

Figure 5 shows the results of application execution in the real environment (the cluster) and its simulation; figure upper side contains the response time graph, for both real and simulated executions, while lower side contains relative percentage error between real timings and the simulation prediction. Note that the application trend in simulated and real environment are the same.

```
<MetaPL>
  <Code>
    <Task name="Gauss"> <Block>
      <Command time="t1" name="readdata"/>
      <Broadcast dim="1024" />
      . . .
    </Block> </Task>
  <Code>
  <Mapping>
  <NumberOfProcesses value="6" />
  <NumberOfGatekeeper value="2" />
  <Gatekeeper id="1">
  <Scheduler="fork">
  </Gatekeeper>
  </Mapping>
</MetaPL>
```

Fig. 4. Gauss-Siedel MetaPL description

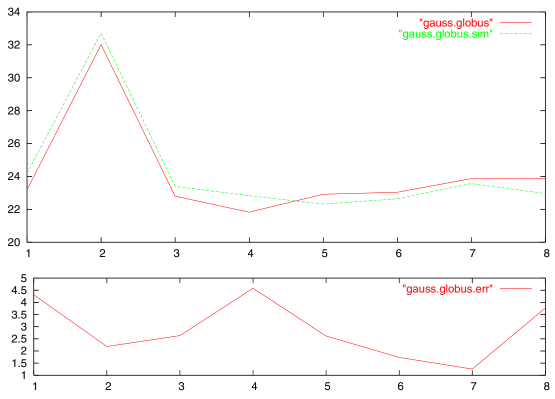


Fig. 5. Gauss-Siedel Simulation

5 Conclusions

This paper shows a performance-oriented approach to GRID application development, founded on the adoption of a simulation environment (HeSSE) and a prototypal language (MetaPL) for performance evaluation of the application at any step of the target software development.

We developed GRID simulation components for the existing simulation environment, able to reproduce the main globus components performance behavior. The proposed components were validated on a test-bed, built upon an SMP cluster.

Then a simple application was developed in the prototypal language (MetaPL), adopting the newly developed GRID extensions, in order to drive the simulations and its performances were predicted.

To validate the approach, we developed the application and compared the results with the predicted ones, showing that the simulated behavior correspond to real system evolution.

Acknowledgment

This work was partially supported by "Centro di Competenze" regione Campania. We want to thank Raffaele Vitolo for his technical support and Rocco Aversa for his contribution.

References

1. I. Foster, C. Kesselman, J. Nick, and S. Tuecke: “The physiology of the grid: An open grid services architecture for distributed systems integration". Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.
2. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. International J. Supercomputer Applications, 15(3), 2001.
3. Buyya, R. , Murshed, M.: “GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", The Journal of Concurrency and Computation: Practice and Experience (CCPE), Wiley Press, May 2002.
4. Takefusa Bricks: A Performance Evaluation System for Scheduling Algorithms on the Grids. JSPS Workshop on Applied Information Technology for Science (JWAITS 2001). 2001.01.
5. Huaxia Xia, Holly Dail, Henri Casanova and Andrew Chien, The MicroGrid: Using Emulation to Predict Application Performance in Diverse Grid Network Environments, In Proceedings of the Workshop on Challenges of Large Applications in Distributed Environments (CLADE ’04), held in conjunction with the Thirteenth IEEE International Symposium on High-Performance Distributed Computing, Honolulu, Hawaii, June 2004 .
6. Henri Casanova and Arnaud Legrand and Loris Marchal, Scheduling Distributed Applications: the SimGrid Simulation Framework,Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid’03)
7. Aversa, R., Mazzeo, A., Mazzocca, N., Villano, U.: Developing Applications for Heterogeneous Computing Environments using Simulation: a Case Study. Parallel Computing 24 (1998) 741-761
8. Mazzocca N., Rak M., Villano U. 2000. “The Transition from a PVM Program Simulator to a Heterogeneous System Simulator: The HeSSE Project". Recent Advances in Parallel Virtual Machine and Message Passing Interface, in J. Dongarra et al. (eds.) Lecture Notes in Computer Science, Vol. 1908, Springer-Verlag, Berlin 2000, (pp. 266-273).
9. N. Mazzocca, M.Rak, R. Torella, E. Mancini and U. Villano, The HeSSE simulation environment. Proc. ESMc’2003, 27-29 Oct. 2003, Naples, Italy, pp. 270-274.
10. Mazzocca N., Rak M., Villano U., 2001. “MetaPL: a Notation System for Parallel Program Description and Performance Analysis" Parallel Computing Technologies, in Malyshkin. V. (ed.), Lecture Notes in Computer Science, Vol. 2127, Springer-Verlag, Berlin 2001,(pp. 80-93)
11. Labarta, J., Girona, S., Pillet, V., Cortes T., Gregoris, L.: DiP: a Parallel Program Development Environment. Proc. Euro-Par ’96, Lyon, France (Aug. 1996) Vol. II 665- 674
12. E. Mancini, N. Mazzocca, M. Rak, and U. Villano, Integrated Tools for Performance-Oriented Distributed Software Development. Proc. SERP’03 Conference, Las Vegas (NE), USA, June 23-26, 2003, vol. I, pp. 88-94

13. N. Mazzocca, M. Rak, U. Villano, The MetaPL approach to the performance analysis of distributed software systems. Proc. 3rd International Workshop on Software and Performance (WOSP02), IEEE Press (2002) 142-149
14. E. Mancini, M. Rak, R. Torella, “U. Villano, Off-line Performance Prediction of Message-Passing Applications on Cluster Systems, Lecture Notes in Computer Science, Vol. 2840, Springer-Verlag, Berlin 2003, (pp. 45-54).

Towards a Bulk-Synchronous Distributed Shared Memory Programming Environment for Grids

Håkan Mattsson[1] and Christoph Kessler[2]

[1] Department of Natural Science and Technology
Gotland University, Sweden
hakan.mattsson@hgo.se
[2] Programming Environments Laboratory (PELAB)
Department of Information and Computer Science
Linköping University, Sweden
chrke@ida.liu.se

Abstract. The current practice in grid programming uses message passing, which unfortunately leads to code that is difficult to understand, debug and optimize. Hence, for grids to become commonly accepted, also as general-purpose parallel computation platforms, suitable parallel programming environments need to be developed.
In this paper we propose an approach to realize a distributed shared memory programming environment for computational grids called *GridNestStep*, by adopting *NestStep*, a structured parallel programming language based on the Bulk Synchronous Parallel model of parallel computation.

1 Introduction

Programming for *grids* [1] is currently not an easy task which is partly due to the distributed memory view that grids provide, leaving the programmer with the burden of handling communication between grid nodes explicitly.

Today, programming for the grid often involves using message passing, for example MPICH-G2 [2], on top of a grid middleware (like Globus Toolkit [3]). As stated in [1] MPICH-G2, Globus Toolkit and other tools "... make it *possible* to write Grid programs, they do not make it *easy*." Much work is still left to the programmer, such as handling communication, synchronization, and load balancing.

In order to ease the burden of programming for a grid environment, alternative programming models are needed. One such model is *distributed shared memory*, in which the programmer sees the grid environment as a single (virtual) parallel computer with a single (virtual) shared memory. Providing this shared view of memory would, ideally, make programming for the grid as easy as for a usual shared memory computer, avoiding the error-prone use of message passing. To achieve this, either the application must be translated into a program utilizing the message passing model, or the message passing is done within an underlying system software layer that emulates a shared memory.

Currently, the most suitable kind of applications for grids are *embarrassingly parallel*, such as parameter studies that simultaneously run multiple copies of the same sequential program, each with different input parameters. Another example application, although in this case in a *peer-to-peer computing* (P2PC) environment, is the `SETI@Home` project (Search for Extraterrestrial Intelligence) [4], where workpackages, containing data, can be downloaded and processed by voluntary participants. However, applications like parameter studies or `SETI@Home` can be considered being of a trivial kind of parallelism, since the workpackages are uniformly structured, no communication is required between the participating grid nodes, and inter-process coordination is limited to dispatching jobs to the grid and collecting the results returned. For applications with a less trivial structure of parallelism, generation of efficient, scalable code for grid and peer-to-peer computing systems is still an unsolved problem.

This research aims at the development of a high-level programming environment called *GridNestStep*, which will provide a distributed shared memory layer on top of a computational grid system. Processor coordination, shared memory consistency model, and communication structure for remote memory accesses in GridNestStep follow the bulk-synchronous parallel (BSP) model of computation. GridNestStep will be built upon previous work by adopting the structured parallel programming language *NestStep* [5,6].

2 The BSP Model and NestStep

NestStep is a parallel programming language for the *Bulk Synchronous Parallel* BSP [7] model of parallel computation. Adopting a single program, multiple data (SPMD) execution style, BSP organizes a parallel computation of a group of p processors into *supersteps* that are separated by barrier synchronizations, see Figure 1.

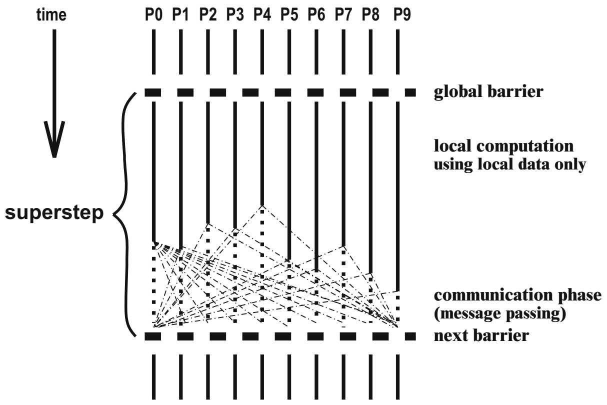


Fig. 1. A BSP superstep

Each superstep consists of a first phase in which a processor performs computation on its local data and locally cached copies of remote data, and a second phase where the processor sends requests for reading or updating remote data. After performing a (conceptual) group-wide barrier synchronization the processor handles the messages received during the previous superstep and proceeds into the next superstep.

Originally the BSP model defined communication in terms of explicit message passing between processors. In contrast, the BSP-based programming language NestStep

[5,6] was designed to support software emulation of shared variables for the BSP model. NestStep also supports nested parallelism by providing language constructs for static and dynamic nesting of supersteps and for performing synchronization of processor subsets. This is done by organizing processors into *groups*, which can be dynamically divided and restored during the execution of a program, thus following the (nested) structure of the supersteps.

NestStep is defined as a set of language constructs that can extend any imperative programming language, as long as unrestricted pointers are avoided. NestStep is implemented as a precompiler that translates the NestStep constructs into the basis language with calls to a runtime system. Then the basis language compiler is used to compile these source files and link them to the NestStep runtime library. Currently supported languages are Java, using the Java API for TCP/IP socket communication, and C, using MPI for communication.

In NestStep, variables, arrays and objects are either *private* to a process or *shared* between a group of processes. For shared variables, arrays and objects, NestStep guarantees that at superstep boundaries all copies contain the same value (*superstep consistency*). NestStep defines two variants of sharing:

– A *replicated* shared variable, array or object exists as a local copy on *each* processor in the group declaring it. Write accesses to the local copy are automatically committed to all remote copies at the end of the superstep.
– A *distributed* shared array is partitioned between the processors in the group, and each processor has direct access to its partition. Read accesses to remote elements are done by prefetching in the preceding superstep's communication phase. Remote write accesses translate to update messages that are committed at the end of a superstep.

In both cases, message passing for updating remote copies of shared variables and array elements occurs *only* in the communication phase at the end of a superstep, which follows the BSP model.

NestStep denotes BSP supersteps by the `step` and `neststep` statements. `step` {statements} groups a set of statements to be executed as one superstep (see Figure 1), while `neststep` (...) {statements} partitions the current processor group into a user-defined number of subgroups such that each subgroup executes the statements independently from the others, as a superstep. For example, the statement `neststep(2,...)` splits the current processor group, say G, into two separate groups G_1 and G_2. After the split, the two processor groups execute their respective supersteps independently of each other, such that changes to shared variables are only committed within each subgroup. Finally, when the subgroups have executed their supersteps, the parent group is restored and subgroup changes to shared variables defined in the parent group, are committed. This supports statically and dynamically nested parallelism and immediately maps to parallel divide-and-conquer computations.

Summarizing, the `step` and `neststep` statements maintain two kinds of invariants during the execution of a NestStep program:

– *Superstep synchronicity*: all processors of the same (active) processor group are working on the same superstep. This implies an implicit groupwide barrier synchronization at the beginning and the end of statement.

– *Superstep consistency*: guarantees that, before a `(nest)step` statement is performed, all processors belonging to the same group have the same value of their local copies of a shared variable.

At the end of a `(nest)step` statement, a *combine* phase is carried out to combine and commit changes made to copies of replicated shared variables and remote shared array elements to reestablish superstep consistency. Communication between processors in a group is structured as a tree. During the combine-phase, messages are sent from the leaf processors towards the root of the tree. The result of the combine phase (where several global operations such as reductions and parallel prefix can be performed on-the-fly), is then committed to the processors of the group by sending messages in the opposite direction.

3 Mapping the BSP Model to a Grid Environment

We propose to adapt the BSP model with the NestStep language to grid computing environments.

As Figure 2 proposes, this could be done e.g. for NestStep-C, that is, using C/C++ with NestStep language extensions. A precompiler translates the NestStep constructs to run-time system calls. The result is an explicitly parallel program decomposed into supersteps. For simplicity, we consider for now only flat (non-nested) supersteps.

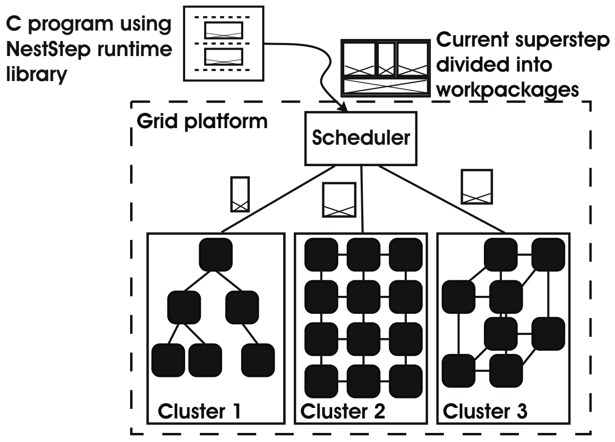

C program using NestStep runtime library
Current superstep divided into workpackages
Grid platform
Scheduler
Cluster 1
Cluster 2
Cluster 3

Fig. 2. Running NestStep programs on a grid system

The (virtual) BSP processors' computations *within* a single superstep are embarassingly parallel, which perfectly matches the constraints encountered in a grid environment. Moreover, by superstep synchronicity, all grid processors involved will, at any time, work on the same superstep. Hence, we can define a *workpackage* by the computation of a single superstep and a set of (virtual) BSP processors to execute that superstep.

Following the hierarchical structure of grids, we partition the computation work of a superstep into workpackages of appropriate size. Within each superstep S we define *macro-workpackages* w_1^S,..., w_P^S by aggregating the work (program code with input data sets and a set of global BSP processor indices) for P disjoint processor subsets to be allocated for execution of S, which are to be farmed out to the P grid nodes (the clusters in Figure 2) available for the parallel computation. The sizes of these subsets are to be chosen depending on the current load of the grid nodes (if known), otherwise load balancing must be handled by the grid middleware. Assuming that each grid node j gets one macro-workpackage w_j^S, then w_j^S can be locally decomposed into smaller workpackages, one for each available processor of j, which are then executed locally, thereby exploiting more fine-grained parallelism within the grid nodes.

After partitioning, the macro-workpackages are sent by a scheduler via the grid middleware to the grid nodes, see Figure 2. The results of executing the macro-workpackages are delivered via the scheduler to the client (NestStep) program.

As mentioned in Section 2, NestStep can define replicated shared variables. Since the value of each of these variables must be identical on all processors of the current group at the beginning of each superstep, this means that at the end of each superstep the processors must communicate their locally held value of the shared variable. So, one problem is how to realize the communication in the grid environment efficiently, and to assure that this phase does not take an arbitrary amount of time, which could happen if some of the processors are still busy for a while with other work before processing their workpackage, thus delaying the entire superstep computation.

In order to combine write accesses to shared variables, and broadcast updated values to all processors of a group at the beginning of the next superstep, we apply the same principle of combine trees as implemented in the NestStep runtime system [5], where the topmost level of the combine tree, including the root (client), spans the different grid nodes and the subtree for each grid node spans cluster-local processors only. Note that these trees are reconstructed for each superstep as the number of available processors per cluster may change.

In the ideal case, the compiled code for the GridNestStep application program is already available on each processor on each grid node, such that a workpackage simply contains a reference to the current superstep and the range of (virtual) BSP processors assigned to each grid node. Note that a physical processor may emulate several (virtual) BSP processors if necessary.

Otherwise, we also need to ship the superstep code along with the workpackage to the grid nodes. In grids with heterogenous processor hardware, this involves the problem of distributing the application in different versions, compiled for the respective node. One solution might be to precompile the application into a library of binary codes, for the different computer architectures. However, this means we will only support a fixed set of architectures. Another possibility is to ship ”raw” source code and compile it at the nodes. This would be a very flexible solution, since we can use any computational resources available at run time, but it requires a compiler along with all required libraries at each node and adds the overhead of compilation time to the time for processing the data. A variant of this scenario ships target-independent byte code that is compiled by a Just-In-Time compiler at the nodes. This has the advantage that we can perform optimization on the code during run time. A further problem is the security aspect, i.e. the prevention of distributing malicious code. However, at this stage we are not concerned with this problem.

The most appealing problem is how to perform load balancing in the system. As nodes in the grid are allocated work, we need a way to monitor the load on each of them, in order to prohibit unbalanced work. An idea is to communicate status information to the scheduler along with the computational results from each node. This information is then analyzed to make load balancing decisions for the next superstep. An alternative could be decentralized load balancing. For instance, a two-tier system with local load balancers on each grid node involved may primarily focus on intra-cluster load balancing but allow for inter-cluster task migration where an entire cluster runs out of local work. Cluster-aware load balancing strategies for multi-cluster environments have been investigated by van Nieuwpoort et al. [8], albeit for more fine-grained and non-BSP computations.

In computational grids with low reliability, as in P2PC systems, there arises the question what should happen when a node, for some reason, becomes unavailable during the processing of a superstep. This can be handled (precisely as in `SETI@Home`) by giving out multiple copies of the same workpackage to several grid nodes and tracking incoming results such that, especially in the case of accumulative updates to shared variables (reductions), only one of the contributions is considered.

4 Towards a Prototype Implementation

Currently we are working on the first steps towards an implementation of GridNestStep by adapting a single-cluster implementation of the NestStep run-time system on top of MPI, and adding further system components (partitioner, scheduler, precompiler) needed for a grid environment. As grid platform we will use the SweGrid hardware of the current Swedish grid initiative [10] together with the NorduGrid middleware [11]. In this case the scheduler resides on the client machine, too. When jobs are submitted, the Nordugrid middleware will make sure that the receiving cluster will match the application-specific requirements on, for example, processor type and availability, memory size, operating system, and installed software. These requirements can be specified by the user in xRSL (Extended Resource Specification Language) [11].

5 Application: GridModelica

GridModelica is a project aiming at creating a grid-enabled extension and implementation of the modeling language Modelica [9]. Modelica is an object-oriented, equation-based language for modeling physical systems. From a model specification, the Modelica compiler generates a large system of ordinary and partial differential equations that is submitted to an iterative solver. Executing the solver corresponds to a simulation of the system. There exist parallel algorithms for such iterative solver algorithms that have a bulk-synchronous parallel structure and thus can be formulated as NestStep programs.

The GridModelica project builds a two-tier system for grid-based multi-domain modeling and simulation, by extending the Modelica language with additional constructs for distributed model component libraries and grid-enabled model composition. At the simulation layer, the GridNestStep programming environment will constitute the computational platform for executing the simulation programs generated by the GridModelica compiler.

6 Summary

Grid computing has become an area of extensive research. However, currently grid programming is error-prone and much work is still left to the programmer. One (partial) solution to this problem is the introduction of a distributed shared memory environment with a simple, grid-compliant shared memory consistency model. We have proposed a language-based approach that follows the bulk-synchronous parallel model, and sketched how the superstep structure of BSP programs can be mapped to computational grids.

Beyond conceptual work on program transformations, communication optimization, load balancing, and workpackage layout, several implementation aspects need further work. We will investigate different ways for the scheduler to get and exploit status information from the grid middleware. In the first implementation we will only be concerned with handling of flat supersteps. However, later on we would like to investigate the possibility to express nested parallelism in GridNestStep and exploit intra-cluster locality by mapping entire processor subgroups to the same grid node.

Acknowledgements

This work is partly supported by VINNOVA, project GridModelica.

References

1. Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann Publishers, second edition, 2003.
2. Nicholas T. Karonis and Brian Toonen and Ian Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63(5):551 – 563, May 2003.
3. The Globus Alliance. http://www.globus.org
4. David P. Anderson and Jeff Cobb and Eric Korpela and Matt Lebofsky and Dan Werthimer SETI@Home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56 – 61, 2002.
5. C. Kessler. Managing distributed shared arrays in a bulk-synchronous parallel programming environment. *Concurrency and Computation: Practice and Experience*, 16:133 – 153, 2004.
6. C. W. Keßler. *NestStep*: Nested Parallelism and Virtual Shared Memory for the BSP Model. *The Journal of Supercomputing*, 17(3):245 – 262, November 2000.
7. L. G. Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, 33(8):103 – 111, August, 1990.
8. R. V. van Nieuwpoort, T. Kielmann and H. E. Bal. Efficient Load Balancing for Wide-Area Divide-and-Conquer Applications., *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, 2001.

9. Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.
10. SweGrid. http://www.swegrid.se
11. P. Eerola et al. The NorduGrid architecture and tools. *Proceedings of Computing in High Energy and Nuclear Physics*, 2003.

High-Performance Computing in Earth- and Space-Science: An Introduction

Organizer: Peter Messmer

Tech-X Corporation
5621 Arapahoe Avenue, Suite A
Boulder, CO 80303, USA
messmer@txcorp.com

1 Introduction

The major goals of earth- and space science investigations (ESS) are the observation, understanding, modeling and prediction of effects on earth and in space. Advanced computing is required on all these stages: The increasing amount of data recorded by sensors has to be processed, archived, and retrieved. In order to extract knowledge from these observations, advanced algorithms are required to assist the scientist in filtering this data in a multitude of ways. Finally, in order to develop predictive capabilities, models are required that can capture all the important effects in a physical system.

Early modeling efforts were mainly driven by effects discovered in new observations. The algorithm were then developed to include the necessary physics to understand these observations. However, the physical systems under investigation in ESS involve typically a multitude of effects on a variety of scales. Modern algorithms therefore try to span broader ranges of scales and capture a multitude of physical processes. This will finally lead to predictive capabilities, one of the goals for e.g. space-weather simulations sponsored by NASA's Sun-Earth Connection Division.

However, the choice of the most appropriate algorithms is only the first step toward successful large scale models. The codes have to be carefully designed to utilize the sophisticated hardware they run on. This can be taken into account relatively easy when designing a new code from scratch. But often large scale codes originate from legacy single effect models. Combining these codes into multi-physics models requires the choice of appropriate data structures to avoid excessive data transformation. And finally, advanced tools are needed for the analysis, reduction and management of the large data sets generated by these codes.

Goal of this mini-symposiums was to bring together physicists and computer-scientists, presenting both computational problems faced by the ESS communities, as well as solutions and tools from high-performance computing.

2 Applications

Over the past few decades, a multitude of algorithms have been developed to model physical processes at various scales involved in ESS models. The assumptions underlying each of these models prove useful in their particular domain and can yield important scientific results. In this mini-symposium, results of several of these models were presented.

Monte-Carlo based methods are an attractive means to investigate phenomena of a statistical nature, like diffusive processes. In order to obtain physically meaningful results, a good sampling of the initial conditions and the configuration space is required. The wide availability of PC-clusters enables these simulations without the extensive costs and complications of High-Performance Computing platforms.

First principles studies are an additional field that draws a lot of attention, especially due to the simplicity of the physical assumptions. These codes usually yield good parallel performance via domain decomposition, allowing them to work efficiently on large scale systems. Increasing computing performance therefore enables first principles models on scales which were otherwise the domain of more approximative methods. An example are fully kinetic electromagnetic models of dusty plasmas, which are usually treated with hybrid kinetic-fluid algorithms.

Nevertheless, for most problems in ESS, first principle models are still computationally too expensive. Often, all the details of the kinetic development are not required and a fluid approximation is valid. Studies like the interaction of the solar wind with a comet or a planet are examples where fluid or MHD models can be very helpful tools.

Finally, for a variety of problems, hybrid approaches can provide the physical fidelity of kinetic models while keeping the speed benefit of fluid models. E.g. for reactive flows, the chemical processes can be modeled using kinetic equations, while the motion is based on fluid equations. The fast execution of these codes and high-performance computing techniques allow modeling such processes faster than in real time.

3 Algorithms and Tools

While many exciting results can be generated using well established algorithms, a lot of models are still limited by present day computing resources. E.g. particle based first principles models are often computationally too demanding. Especially constraints for numerical stability, like the Courant condition, prohibit large scale simulations over extended periods of time. New algorithms, sometimes based on fundamentally new approaches, are therefore required to relax those constraints.

Improved algorithms are the most advantageous approach to decrease the computational cost. However, these algorithms tend to be complex by themselves, and their parallel implementation can add an unnecessary burden on the developers. Tools are therefore required that simplify these implementations. This is particularly true for multi-physics models. Most of these models were developed as standalone applications and in order to couple them, an infrastructure is required to mediate between the quantities of the different models. E.g. to model magnetic reconnection, it would be beneficial to model the actual reconnection region in a fully kinetic way, whereas in the outer regions fluid or hybrid approximations are sufficient. Another example are global climate models, where oceanic and atmospheric models have to be coupled and quantities have to be converted between the different codes. Providing an infrastructure for these tasks is therefore highly desirable.

4 Conclusion

The mini-symposium on High-Performance Computing in Earth and Space-Science brought together an exciting mix of papers from both physics and computer science. It became clear that cooperation between these communities is important in order to fully exploit the potential of present-day high-performance computing platforms. I hereby wish to thank the PARA04 organizers for hosting this mini-symposium, all the referees for reviewing the contributed papers and finally all the authors who submitted their papers to the workshop.

Applying High Performance Computing Techniques in Astrophysics*

Francisco Almeida[1], Evencio Mediavilla[2], Alex Oscoz[2], and Francisco de Sande[1]

[1] Dept. de Estadística, Investigación Operativa y Computación
Univ. de La Laguna, 38271–La Laguna, Spain
{falmeida,fsande}@ull.es
[2] Instituto de Astrofísica de Canarias (IAC), c/Vía Láctea s/n, 38271–La Laguna, Spain
{emg,aoscoz}@ll.iac.es

Abstract. When trying to improve the execution time of scientific applications using parallelism, two alternatives appear as the most extended: to use explicit message-passing or using a shared address memory space. MPI and OpenMP are nowadays the industrial standards for each alternative. We broach the parallelization of an astrophysics code used to measure various properties of the accretion disk in a black hole. Different parallel approaches have been implemented: pure MPI and OpenMP versions using cyclic and block distributions, a hybrid MPI+OpenMP parallelization and a MPI Master-Slave strategy. A broad computational experience on a ccNUMA SGI Origin 3000 architecture is presented. From the scientific point of view, the most profitable conclusion is the confirmation of the robustness of the technique that the original code implements.

1 Introduction

This work collects the experiences of a collaboration between researchers coming from two different fields: astrophysics and parallel computing. We deal with different approaches to the parallelization of a scientific code that solves an important problem in astrophysics, the detection of supermassive black holes. To accomplish it, light curves of Quasi-stellar Object (QSO) images must be analytically modeled. The scientific aim is to find the values that minimize the error between this theoretical model and the observational data according to a chi-square criterion. The robustness of this procedure depends on the sampling size m. However, even for relatively small values of m, the determination of the minimum takes a long time of execution since m^5 starting points of the grid must be considered, as will be shown later in section 2. We will show that parallelization can reduce the total time of execution, allowing to greatly increase the sampling and, consequently, to improve the robustness of the evaluated minimum.

The sequential code is best suited for different parallel implementations varying from message-passing (MPI) to shared-memory (OpenMP) codes and hybrid solutions combining both (MPI+OpenMP). One of the aims of this work is to obtain the maximum reduction in the execution time for the original sequential code. With this goal in mind,

* This work has been partially supported by the EC (FEDER) and the Spanish MCyT (Plan Nacional de I+D+I, TIC2002-04498-C05-05 and TIC2002-04400-C03-03).

we have also to consider that the amount of time invested in code developments is a heavy limitation for the usual non-parallel expert scientific programmers. To balance both objectives we devise the necessary efforts comparing pure MPI and OpenMP codes and mixed MPI+OpenMP programs also. The parallel approaches have been tested and a broad computational experience on a ccNUMA SGI Origin 3000 will be presented. The conclusions and the knowledge acquired with our experiments establish the basis for future developments on similar codes. The paper has been structured as follows. Section 2 describes the astrophysics problem and the code to be parallelized. This sequential code is slightly modified to facilitate the four parallel developments introduced in section 3. The computational experience is presented in section 4. We conclude that, for the application considered, no significative differences have been found among the performances of the different parallelizations. However, the development effort of the considered versions could be appreciable. We finalize the paper in section 5 with some concluding remarks and future lines of work.

2 The Problem

Supermassive black holes (SMBH: objects with masses in the range $10^7 - 10^9$ solar masses) are supposed to exist in the nucleus of many if not all the galaxies. It is also assumed that some of these objects are surrounded by a disk of material continuously spiraling (accretion disk) towards the deep gravitational potential pit of the SMBH and releasing huge quantities of energy giving rise to the phenomena known as quasars.

We are interested in objects of dimensions comparable to the Solar System in galaxies very far away from the Milky Way. Objects of this size can not be directly imaged and alternative observational methods are used to study their structure. One of these methods is the observation of Quasistellar Object (QSO) images affected by a microlensing event to study the unresolved structure of the accretion disk. If light from a QSO pass through a galaxy located between the QSO and the observer it is possible that a star in the intervening galaxy crosses the QSO light beam. Thus the gravitational field of the star can amplify the light emission coming from the accretion disk (gravitational microlensing). As the star is moving with respect to the QSO light beam, the amplification varies during the crossing. The curve representing the change in luminosity of the QSO with time (QSO light curve) depends on the position of the star and on the structure of the accretion disk. The goal in this work is to model light curves of QSO images affected by a microlensing event to study the unresolved structure of the accretion disk.

According to this objective we have fitted the peak of an high magnification microlensing event recently observed in one quadruple imaged quasar, Q 2237+0305. We have modeled the light curve corresponding to an standard accretion disk [5] amplified by a microlens crossing the image of the accretion disk. Leaving aside the physical meaning of the different variables (for details see [2]), the function modeling the dependence of the observed flux, F_ν, with time, t, can be written as

$$F_\nu(t) = A_\nu \int_1^{\xi_{max}} \{1 + \frac{B}{\sqrt{\xi}} G[C(t-t_0)/\xi]\} \frac{\xi d\xi}{e^{(D_\nu \xi^{3/4}(1-1/\sqrt{\xi})^{-1/4})} - 1} \tag{2.1}$$

where ξ_{max} is the ratio between the outer and inner radii of the accretion disc (we will adopt $\xi_{max} = 100$). G is a function

$$G(q) = \int_{-1}^{+1} \frac{H(q-y)dy}{\sqrt{q-y}\sqrt{1-y^2}},$$

To speed up the computation, G has been approximated by using MATHEMATICA (for details, see appendix in [2]).

Therefore, the goal is to estimate the values of the parameters A_ν, B, C, D_ν, and t_0 by fitting F_ν to the observational data. Specifically, to find those values of the parameters that minimize

$$\chi^2 = \sum_{i}^{N} \frac{(F_i^{obs} - F_\nu(t_i))^2}{{\sigma_i}^2} \tag{2.2}$$

where N is the number of data points (F_i^{obs}) corresponding to times t_i $(i = 1, ..., N)$ and $F_\nu(t_i)$ is the theoretical function evaluated at t_i. σ_i is the observational error associated to each data value. To minimize χ^2 we used the NAG [4] routine `e04ccf`. This routine [3] only requires evaluation of the function and not of the derivatives. As the determination of the minimum in the 5-parameters space depends on the initial conditions we needed to consider a 5-dimensional grid of starting points. If we consider m sampling intervals in each variable, the number of starting points in the grid is of m^5. For each one of the points of the grid we computed a local minimum. Finally, we select the absolute minimum among them. Even for relatively small values of m, the determination of the minimum takes a long time of execution.

Figure 1 shows the original sequential version of the code. There are five nested loops corresponding to the traverse of the 5-dimensional grid of starting points. We minimize the `jic2` function using the `e04ccf` NAG function. The starting point `x`, and the evaluation of `jic2(x)` (`fx`) are supplied as input/output parameters to the NAG

```
      program seq_black_hole
      double precision t2(100), s(100), ne(100), fa(100), efa(100)
      common/data/t2, fa, efa, length
      double precision t, fit(5)
      common/var/t, fit

c     Data input
c     Initialize best solution
      do k1=1, m
        do k2=1, m
          do k3=1, m
            do k4=1, m
              do k5=1, m
c             Initialize starting point x(1), ..., x(5)
              call jic2(nfit,x,fx)
              call e04ccf(nfit,x,fx,ftol,niw,w1,w2,w3,w4,w5,w6,jic2,
     x                    monit,maxcal,ifail)
              if (fx improves best fx) then
                update(best (x, fx))
              endif
              enddo
            enddo
          enddo
        enddo
      enddo
```

Fig. 1. Original sequential pseudocode

function. Inside the `jic2` function, the integral of formula 2.1 is computed using the `d01ajf` and `d01alf` NAG functions. These functions are general-purpose integrators that calculate an approximation to the integral of a function. The common blocks in lines 3 and 5 are used to pass additional parameters from the `jic2` function to these integrators. Observe that `jic2` is passed as a parameter to the minimization function `e04ccf`.

3 Parallelization

The parallelization was coerced by two main constraints: to introduce the minimum amount of changes in the original code and to preserve the use of the NAG functions. At the beginning of our collaboration, one objective was that the kind of parallelization to be introduced in the code should be easily understood and reproduced by other non parallel programmers.

The only modification introduced in the original sequential version was an ad hoc transformation of the iteration space by reducing the five nested loops in listing 1 to a single one. This modification does not change the underlying semantics of the sequential code nor the order in which the points are evaluated. The transformation can be easily achieved since there are no data dependencies among the different points in the iteration domain. The reason to introduce this transformation is to ease and clarify the parallelizations.

MPI and OpenMP pure versions can be directly obtained from that code. The MPI parallelization is the most straightforward: after inserting in the code the initialization and finalization MPI calls, we simply divide the single loop iteration space among the available set of processors using a cyclic or block distribution. In the pure OpenMP parallelization, the main difficulty is to identify the shared/private variables in the code. Once they are located, the corresponding OpenMP `parallel do` pragma can be introduced in the main loop. Variables included in `common` blocks deserved a particular consideration. To avoid read/write concurrent accesses, the `threadprivate` pragma was included. The mixed MPI/OpenMP version just merges both former versions. Each MPI process expands a certain number of OpenMP threads that take in charge the iteration chunk of the MPI process. As it will be shown in the next section, none of the different versions present a linear speedup. This was the motivation to introduce a MPI Master-Slave approach. This alternative has the disadvantage of requiring a higher expertise in parallel programming and therefore it was introduced only to correct the load imbalance of the previous versions.

4 Computational Results

This section is devoted to investigate and quantify the performance obtained with the different parallel approaches taken. On a Sun Blade 100 Workstation running Solaris 5.0 and using the native Fortran77 Sun compiler (v. 5.0) with full optimizations the code takes 5.89 hours and 12.45 hours for sampling intervals of size $m = 4$ and $m = 5$ respectively.

Table 1. Time for parallel codes, sampling intervals: $m = 4$

	Time (secs.)				
P	B-MPI	B-OMP	C-MPI	MPI-OMP	MS-MPI
2	1652.34	1670.51	1674.85	1671.59	1641.94
4	923.40	930.64	951.12	847.37	821.64
8	481.81	485.69	496.05	481.89	412.44
16	256.14	257.20	256.95	255.27	211.54
32	133.00	133.97	133.87	133.70	113.74

Table 2. Time for the mixed (MPI-OpenMP) code for different combinations Threads/MPI Processes. Sampling intervals: $m = 4$ The number of MPI processes is $\frac{P}{nTh}$

	P				
nTh	32	16	8	4	2
1	**133.70**	257.55	497.12	953.34	1679.17
2	137.80	262.26	**481.89**	**847.37**	**1671.59**
4	151.84	278.10	485.05	938.15	-
8	156.56	**255.27**	484.77	-	-
16	139.03	256.34	-	-	-
32	133.88	-	-	-	-

The target architecture for all the parallel executions has been a Silicon Graphics Origin 3000 with 128 R14000 (600Mhz) processors. Only 32 processors were used in the computational experiments. We restricted our executions to this architecture because it is the only available to us with the NAG libraries installed.

Using the native MIPSpro Fortran77 compiler (v. 7.4) and the NAG Fortran Library - Mark 19 in the SGI Origin 3000 with full optimizations the sequential running time is 0.91 hours and 2.74 hours for sampling intervals of size $m = 4$ and $m = 5$ respectively.

Table 1 shows execution time (in seconds) of the parallel codes with sampling intervals of size $m = 4$. Labels B-MPI and C-MPI correspond to the MPI code with block and cyclic distributions of iterations respectively. The label B-OMP corresponds to a block distribution using OpenMP. The mixed mode code (label MPI-OMP) correspond to the minimum times obtained for different combinations of MPI processes/OpenMP threads (see table 2). Column labeled MS-MPI shows the time corresponding to the MPI Master-Slave parallel version. Assuming that we do not have exclusive mode access to the architecture, the times collected in all the cases correspond to the minimum time from five different executions. Except for the Master-Slave version, times do not show a significative variation among the different parallel approaches taken.

The speedups achieved reduce the sequential running time almost linearly with the number of processors involved. In Figure 2 we observe that in both MPI and OpenMP pure versions (C-MPI and B-OMP labels respectively), as the number of processors increases, the growth of the speedup diminishes (the MPI and OpenMP speedup curves appear overlapped in the figure). The reason for this behaviour is the call to the `e04ccf`

NAG routine inside the loop. The convergence run time for this routine is different depending on the starting point. This fact introduces load imbalance since each iteration may consume different run time and some processors may be overloaded. For each processor, figure 3 shows the run time consumed in the loop in an execution with 32 processors. We observe a large variation in the time for MPI and OpenMP versions as a consequence of the imbalance, while the Master-Slave (MS label in figure 2) code shows almost constant time for each processor. In the case of OpenMP, we have also tried to use dynamic scheduling strategies. Figure 4 compares the load imbalance introduced by this scheme against the Master-Slave balanced approach. We consider that with the current implementation of the OpenMP compiler, the overhead introduced with the dynamic scheduling is still unaffordable.

Table 3. Time and speedup for parallel codes, sampling intervals: $m = 5$

	Time (secs.)				Speedup			
P	MPI	OMP	MPI-OMP	MS	MPI	OMP	MPI-OMP	MS
2	4957.0	5085.9	4973.1	4906.4	2.0	1.9	2.0	2.0
4	2504.9	2614.5	2513.0	2455.0	3.9	3.8	3.9	4.0
8	1261.2	1372.4	1265.1	1231.5	7.8	7.2	7.8	8.0
16	640.3	755.8	642.9	619.3	15.4	13.1	15.4	15.9
32	338.1	400.1	339.2	325.4	29.2	24.7	29.1	30.3

For the mixed MPI/OpenMP parallel version, we have considered in the computational experiment different combinations between MPI processes and OpenMP threads. Table 2 shows the running time of this parallel version. An execution with P processors involves nTh OpenMP threads and $\frac{P}{nTh}$ MPI processes. The minimum running times appear in boldface. As it could be expected, the execution time using P MPI processes and 1 OpenMP thread coincide with the correspondent MPI pure version. Analogously, the same behaviour is observed with 1 MPI process and P threads with respect to the pure OpenMP version. A regular pattern for the optimal combination of MPI processes and OpenMP threads is not observed.

Table 3 collects the time and speedup corresponding to sampling intervals of size $m = 5$. We observe no significative variations among the different versions. This effect is due to a better load balance produced by the increment in the iteration space size. Almost linear speedup is observed in all the cases. The lowest speedup values are obtained with the OpenMP version, and again no significative improvement is achieved with the mixed-mode MPI/OpenMP approach.

5 Conclusions and Future Work

We conclude a first step of cooperation in the way of applying parallel techniques to improve performance in astrophysics codes. The scientific aim of applying high performance computing to computationally-intensive codes in astrophysics has been successfully achieved. The relevance of our results do not come directly from the particular

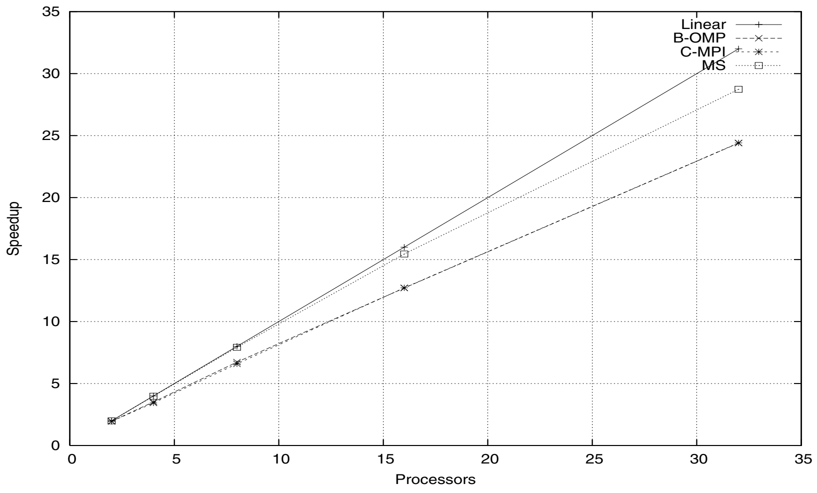


Fig. 2. Speedup achieved by different parallel approaches

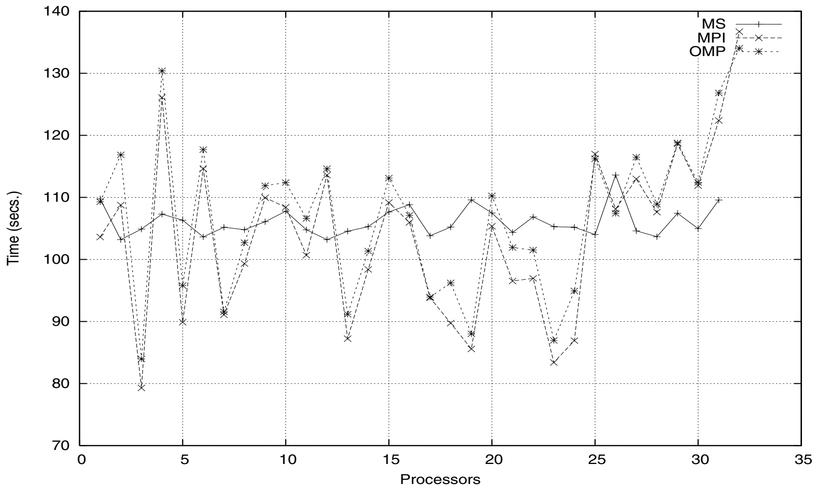


Fig. 3. Load Balance: Master-Slave, MPI and OpenMP

application chosen, but from stating that parallel computing techniques are the key to broach large size real problems in the mentioned scientific field.

For the case of non-expert users and the kind of codes we have been dealing with, we believe that MPI parallel versions are easier and safer. In the case of OpenMP, the proper usage of data scope attributes for the variables involved may be a handicap for users with non-parallel programming expertise. The higher current portability of the MPI version is another factor to be considered.

The mixed MPI/OpenMP parallel version is more expertise-demanding. Nevertheless, as it has been stated by several authors [1], [6], and even for the case of a hybrid

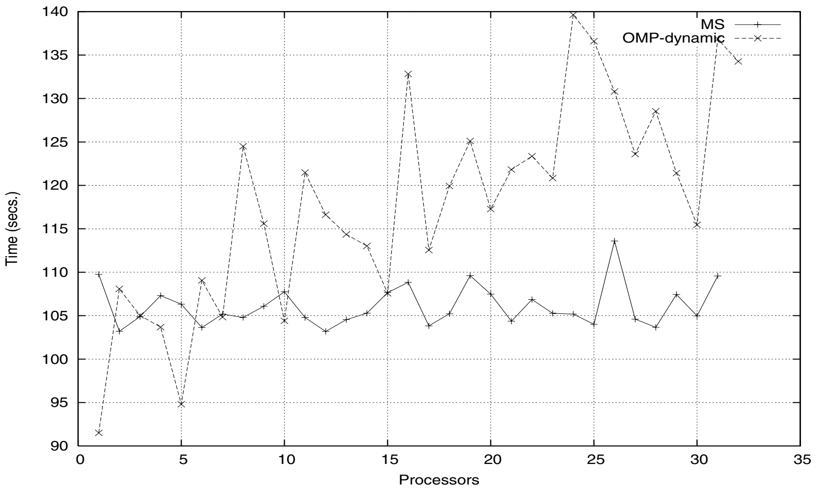


Fig. 4. Load Balance: Master-Slave vs. OpenMP with dynamic schedule

architecture like the SGI Origin 3000, this version does not offer any clear advantage and it has the disadvantage that the combination of processes/threads has to be tuned.

The Master-Slave strategy appears as the most efficient when dealing with load imbalance even in the case of simple codes. This approach provides a satisfactory solution from the parallel programmer point of view, but it requires more parallel programming knowledge, and therefore it is not commendable for non-expert users. OpenMP does not provide easy solutions to broach the load imbalance situation.

From the scientific point of view, the most profitable conclusion is the confirmation of the robustness of the method provided by the computational results.

From now on we plan to continue this fruitful collaboration by applying parallel computing techniques to some other astrophysical challenge problems.

References

1. F. Cappello, D. Etiemble, MPI versus MPI+openMP on IBM SP for the NAS benchmarks, in: Proceedings of Supercomputing'2000 (CD-ROM), IEEE and ACM SIGARCH, Dallas, TX, 2000, lRI.
2. L. J. Goicoechea, D. Alcalde, E. Mediavilla, J. A. Muñoz, Determination of the properties of the central engine in microlensed QSOs, Astronomy and Astrophysics 397 (2003) 517–525.
3. J. A. Nelder, R. Mead, A simplex method for function minimization, The Computer Journal 7 (4) (1965) 308–313.
4. Numerical Algorithms Group, NAG Fortran library manual, mark 19, NAG, Oxford, UK (1999).
5. N. I. Shakura, R. A. Sunyaev, Black holes in binary systems. observational appearance, Astronomy and Astrophysics 24 (1973) 337–355.
6. L. Smith, M. Bull, Development of mixed mode MPI/OpenMP applications, Scientific Programming 9 (2–3) (2001) 83–98.

Statistical Properties of Dissipative MHD Accelerators

Kaspar Arzner[1], Loukas Vlahos[2], Bernard Knaepen[3], and Nicolas Denewet[3]

[1] Paul Scherrer Institut
Laboratory for Astrophysics
CH-5232 Villigen PSI, Switzerland
arzner@astro.phys.ethz.ch

[2] Aristotle University
Institute of Astronomy, Dept. of Physics
54006 Thessaloniki, Greece
vlahos@astro.auth.gr

[3] Université Libre de Bruxelles
Mathematical Physics Dept.
CP231, Boulevard du Triomphe
1050 Bruxelles, Belgium
bknaepen@ulb.ac.be

Abstract. We use exact orbit integration to investigate particle acceleration in a Gauss field proxy of magnetohydrodynamic (MHD) turbulence. Regions where the electric current exceeds a critical threshold are declared to be 'dissipative' and endowed with super-Dreicer electric field $\mathbf{E}_{\Omega} = \eta \mathbf{j}$. In this environment, test particles (electrons) are traced and their acceleration to relativistic energies is studied. As a main result we find that acceleration mostly takes place within the dissipation regions, and that the momentum increments have heavy (non-Gaussian) tails, while the waiting times between the dissipation regions are approximately exponentially distributed with intensity proportional to the particle velocity. No correlation between the momentum increment and the momentum itself is found. Our numerical results suggest an acceleration scenario with ballistic transport between independent 'black box' accelerators.

1 Introduction

Astrophysical high-energy particles manifest as cosmic rays or, indirectly, as radio waves, X-rays, Gamma rays. These often occur in transients, and with distinctly non-equilibrium energy distributions. A prominent source of sporadic radio- and X-ray emission is the Sun during the active phase of its 11-year cycle. Among the numerous mechanisms proposed for accelerating solar particles to high energies (see [1] for an overview), stochastic ones attracted particular attention because they require generic input data and do not rely on special geometrical assumptions. In stochastic acceleration [2,3,4,5], particles move in random electromagnetic fields, where they become repeatedly deflected and, on average, accelerated. The electromagnetic fields are thought to arise from magneto-hydrodynamic (MHD) turbulence (e.g., [6]), perhaps excited by the broadband echo of a magnetic collapse. The turbulence may host shocks and other forms of dissipation if critical velocities [7] or electric current densities [8,9] are exceeded. Associated with dissipation are (collisional or anomalous) resistivity and non-conservative electric fields, which sustain, locally, the electric current against dissipative drag in order to meet the global constraints. However, a detailed balance on the level of individual charge carriers is impossible because the dissipative drag depends on particle position *and* velocity, whereas the electric field is a function of position only. Thus the electric field may compensate the bulk drag, but a (high-energy) population can be left over and exposed to acceleration [10]. This lack of detailed balance is in the heart of dissipative acceleration mechanisms. In plasmas, dissipation occurs at 'ruptures' of the magnetic structure, and is therefore localized around critical points of the magnetic field.

The above scenario, first envisaged by Parker [11] for the solar atmosphere, has since been explored in a large number of numerical studies [12,13,14,6,15,16,17]. On the theoretical side, most stochastic acceleration theories [2,18,19] base on Fokker-Planck approaches, thus transferring two-point functions of the electromagnetic fields into drift and diffusion coefficients of particles by probing the fields along unperturbed trajectories. Dynamical particle averages are then replaced by field ensemble averages, neglecting the fact that real particles move in *one* realization of the random field. As a result, diffusive behaviour may be predicted even if particles are trapped in a single realization of the random field.

In order to investigate the full diversity of orbit behaviour one must resort to numerical simulations. In the present contribution we analyze the behaviour of test particles in resistive MHD turbulence with localized dissipation regions, with particular emphasis on the validity of a Fokker-Planck description [20]. We use here exact orbit integration, and thus avoid any guiding centre approximations [21,22]. The price for rigorosity is computational cost, which makes the scheme only feasible with the aid of high-performance computing.

2 Acceleration Environment

The MHD turbulence has been been modeled by full 3D spectral MHD simulations and by Gauss field proxies [23,17]. We concentrate here on the latter, which is computed from the vector potential $\mathbf{A}(\mathbf{x},t) = \sum_{\mathbf{k}} \mathbf{a}_{\mathbf{k}} \cos(\mathbf{k}\cdot\mathbf{x} - \omega(\mathbf{k})t - \phi_{\mathbf{k}})$ by means of tabulated trigonometric calls. This allows to continuously determine the fields at the exact particle position, and avoids any real-space discretization artifacts, but the computational overhead restricts the $\mathbf{k}$ sum to a few 100 Fourier modes $\mathbf{a}_{\mathbf{k}}$. They are taken from the shell $\min(l_i^{-1}) < |\mathbf{k}| < 10^{-2} r_L^{-1}$ with r_L the rms thermal ion Larmor radius and l_i the outer scale of the power spectral density $\langle |\mathbf{a}_{\mathbf{k}}|^2 \rangle \propto (1 + l_x^2 k_x^2 + l_y^2 k_y^2 + l_z^2 k_z^2)^{-\nu}$. The electromagnetic fields are then obtained from

$$\mathbf{B} = \nabla \times \mathbf{A} \tag{2.1}$$

$$\mathbf{E} = -\partial_t \mathbf{A} + \eta(\mathbf{j})\,\mathbf{j}\,, \tag{2.2}$$

where $\mu_0 \mathbf{j} = \nabla \times \mathbf{B}$ and $\eta(\mathbf{j}) = \eta_0\, \theta(|\mathbf{j}| - j_c)$ is an anomalous resistivity switched on above the critical current $j_c \sim enc_s$. Here, c_s and n are the sound speed and number density of the background plasma. The Gauss field $\mathbf{A}$ must satisfy the MHD constraints

$$\mathbf{E}\cdot\mathbf{B} = 0 \quad \text{if} \quad \eta(\mathbf{j}) = 0 \qquad \text{and} \qquad E/B \sim v_A \tag{2.3}$$

with v_A the Alfvén velocity. Equation (2.3) can be achieved in several ways. For instance, one can use Euler potentials of which only one is time-dependent, or force $\mathbf{A}$ to point along a single direction. A somewhat more flexible way, used here, is axial gauge $\mathbf{a_k} \cdot \mathbf{v}_A$ = 0 with dispersion relation $\omega(\mathbf{k})$ = $\mathbf{k} \cdot \mathbf{v}_A$. A constant magnetic field B_0 along $\mathbf{v}_A$ can be included without violating Eq. (2.3), and we set $|\mathbf{v}_A|^2$ = $(B_0^2 + \sigma_B^2)/(\mu_0 n m_p)$ with σ_B the rms magnetic fluctuations and m_p the proton rest mass. In the present simulation, $\mathbf{v}_A$ and the background magnetic field are along the z direction. The total magnetic field $\mathcal{B}$ = $\sqrt{B_0^2 + \sigma_B^2}$ is a free parameter, which defines the scales of the particle orbits. In order to represent coronal turbulence we choose $v_A \sim 2 \cdot 10^6$ m/s, ν = 1.5, $\mathcal{B} \sim 10^{-2}$T, σ_B = 10^{-2}T, B_0 = 10^{-3}T, and l_x = l_y = 10^3m, l_z = 10^4m. The current threshold j_c is exceeded in about 7% of the total volume. Note that our choice represents strong ($\sigma_B \gg B_0$) and anisotropic ($l_z \gg l_x, l_y$) turbulence. To embed our simulation in the real solar atmosphere one should associate l_z with the radial direction in order to reproduce the predominant orientation of coronal filaments.

3 Particle Dynamics

3.1 Physical Scaling

Time is measured in units of the (non-relativistic) gyro period Ω^{-1}=$m/q\mathcal{B}$; velocity in units of the speed of light; distance in units of $c\Omega^{-1}$. Particle momentum is measured in units of mc; the vector potential in units of mc/q; the magnetic field in units of $\mathcal{B}$; the electric current density in units of $\Omega\mathcal{B}/(\mu_0 c)$, so that the dimensionless threshold current is $j_c' = (m/m_p)c_s c/v_A^2$; and the electric field is measured in units of $c\mathcal{B}$, so that the dimensionless Dreicer [10] field is E_D' = $(v_e'/\tau')(m_e/m)$ with v_e' the electron thermal velocity and τ' the electron-ion collision time. The dimensionless equations of motion are

$$\frac{d\mathbf{x}'}{dt'} = \mathbf{v}' \tag{3.4}$$

$$\frac{d(\gamma\mathbf{v}')}{dt'} = \mathbf{v}' \times \mathbf{B}' - \frac{\partial \mathbf{A}'}{\partial t} + \eta'(|\mathbf{j}'|)\,\mathbf{j}' \tag{3.5}$$

with γ the Lorentz factor, $\mathbf{B}' = \nabla' \times \mathbf{A}'$, and $\mathbf{j}' = \nabla' \times \mathbf{B}'$ the electric current. The dimensionless resistivity η' is characterized by the resulting dissipative electric field $\mathbf{E}_\Omega = \eta_0 \mathbf{j}$ relative to the Dreicer field E_D. We chose η' such that $E_\Omega/E_D \sim 10^4$.

3.2 Particle Initial Conditions

We consider electrons as test particles. The initial positions are uniformly distributed in space, and the velocities are from the tail $v \geq 3\ v_{th}$ of a maxwellian of 10^6 K, which is typical for the solar corona. Coulomb collisions are neglected, which is a good approximation once the acceleration has set on, but is not strictly correct in the beginning of the simulation.

3.3 Numerical Implementation and Simulation Management

Equations (3.4) and (3.5) are integrated by traditional leapfrog and Runge-Kutta schemes. The test particle code is written in FORTRAN 90/95 and compiled by the

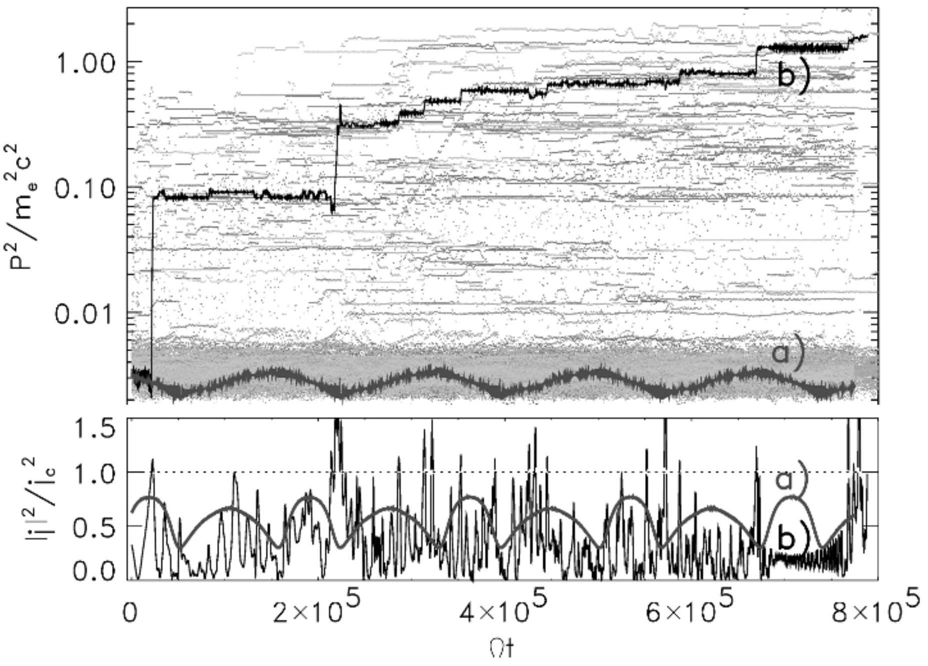


Fig. 1. Evolution of electron kinetic momentum. Top panel: 200 sample orbits; adiabatic (a), and accelerated (b) cases. Bottom panel: electric current density along the orbits a) and b). The critical current density ($|\mathbf{j}| > j_c$) is marked by dotted line. The present simulation is an extension of the simulation of [17]

Portland Group's Fortran 90 compiler (`pgf90`). Diagnostics and visualization uses IDL as a graphical back-end. The code is run on the MERLIN cluster of the Paul Scherrer Institut, and on the ANIC-2 cluster of the Université libre de Bruxelles. The ANIC-2 cluster has 32 single Pentium IV nodes, a total of 48 Gbyte memory, and Ethernet connections. The MERLIN cluster consist of 56 mostly dual Athlon nodes with a total of 80 GByte memory, operated under Linux and connected by Myrinet and Ethernet links. MERLIN jobs are managed by the Load Sharing Facility (LSF) queueing system. Parallelization is done on a low level only, with different (and independent) test particles assigned to different CPU's. MPICH/MPI is used to ensure crosstalk-free file I/O. The field data are computed on each CPU for the actual particle position. Random numbers are needed in the generation of the Fourier amplitudes and -phases of the electromagnetic fields, and in the particle initial data; they are taken from the intrinsic random number generator of `pgf90`.

4 Diagnostics and Results

In order to characterize the relativistic acceleration process we consider the evolution of the kinetic momentum $\mathbf{P}' = \gamma \mathbf{v}'$. This quantity is directly incremented by the equation

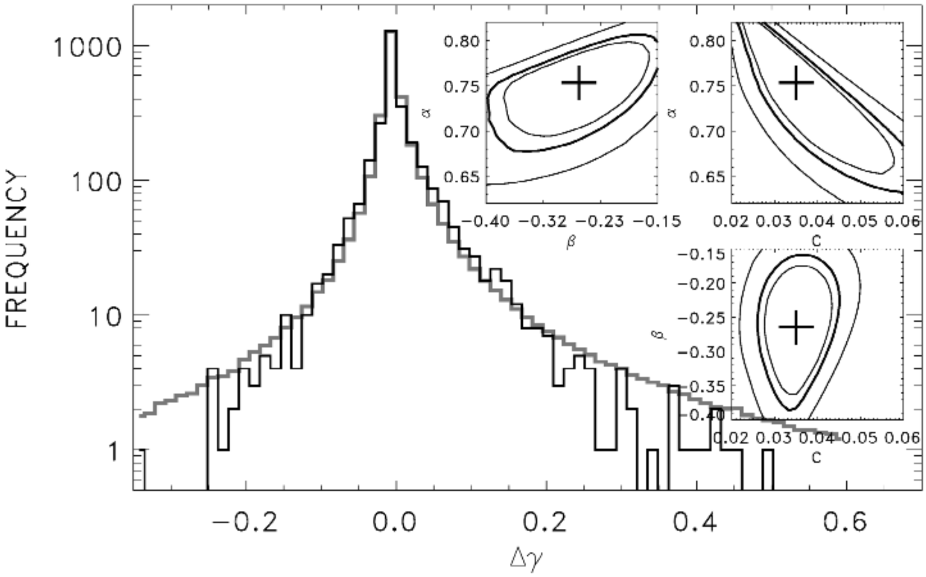
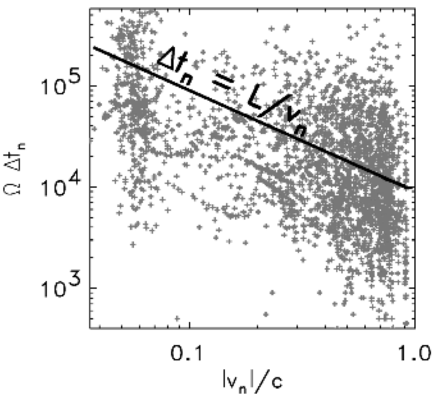


Fig. 2. Frequency distribution of the energy jumps $\Delta\gamma$ of Fig. 1 (black line), together with a best-fit Lévy density (gray line) with parameters $\alpha_0 = 0.75$, $\beta_0 = -0.26$, and $C_0 = 0.035$ (crosses). Inlets: cuts of the likelihood surface at (α_0, β_0, C_0). The 99% confidence level is marked boldface



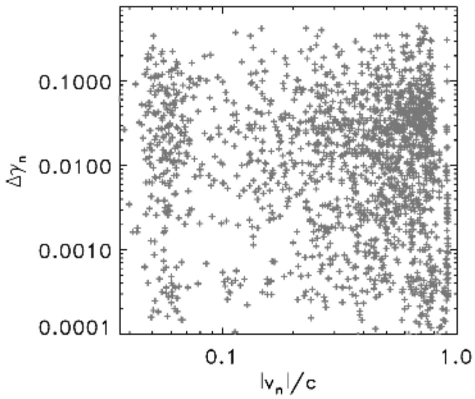


Fig. 3. Left: travel times $\Delta t_n = t_{n+1} - t_n$ between acceleration regions versus velocity v_n. Right: energy gain $\Delta\gamma_n = \gamma_{n+1} - \gamma_n$ versus velocity. While Δt_n scales with inverse velocity (solid line: best-fit), there is no clear trend in the energy gain $\Delta\gamma_n$

of motion (3.5), and – ignoring quantum effects – can grow to arbitrarily large values, so that it can serve as a diagnostics of diffusive behaviour. Alternatively we may use the kinetic energy γ.

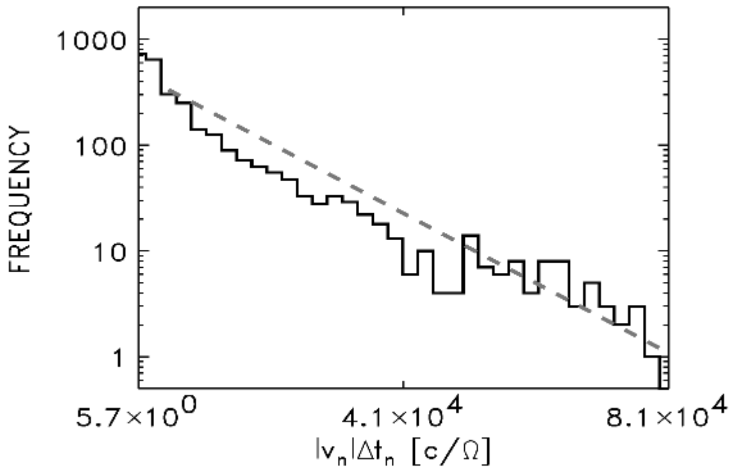


Fig. 4. Histogram of the quantity $|v_n|\varDelta t_n$, together with an exponential fit (dashed)

The results of the orbit simulations are shown in Figs. 1 - 4. When initially superthermal ($v \gtrsim 3v_{th}$) electrons move in the turbulent electromagnetic fields (Eqns. 2.1, 2.2), some of them may become stochastically accelerated. From a population of 600 electrons we find that 35% of the particles are accelerated, while the other 65% remain adiabatic [22,24] during the simulation ($\varOmega t \leq 8{\cdot}10^5$). The two cases are illustrated in Fig. 1 (top). The orbit a) conserves energy adiabatically during the whole simulation, while the orbit b) does not. The orbits of 200 randomly chosen particles are also shown (gray) to trace out the full population. The bottom panel of Fig. 1 shows the the electric current density along the orbits a) and b). Time intervals where the critical current (doted line) is exceeded correspond to visits to the dissipation regions. As can be seen, acceleration (or deceleration) occurs predominantly within the dissipation regions. Accordingly, the orbit b) which never enters a dissipation region remains adiabatic. As a benchmark we have set η_0 = 0 and found that no acceleration takes place at all, thus reproducing the 'injection problem' [25]. Smaller η' yield smaller (than 35%) fractions of accelerated particles.

A glance at graph a) of Fig. 1 shows that $\mathbf{P}'(t')$ is poorly represented by a Brownian motion [20] with continuous sample paths. Rather, $\mathbf{P}'$ changes intermittently and in large jumps. Indeed, if we consider the energy change $\varDelta\gamma_n = \gamma_{n+1} - \gamma_n$ across a dissipation region, we find that its distribution $P(\varDelta\gamma_n)$ has heavy tails and a convex shape which deviates from a Gaussian law (Fig. 2 black line). In order to characterize $P(\varDelta\gamma_n)$ we have tried to fit it by a (skew) Lévy stable distribution $P_L(x)$. The latter is defined in terms of its Fourier transform [26]

$$\phi_L(s) = \exp\Big\{ - C|s|^\alpha \Big(1 + i\beta\frac{s}{|s|}\tan\frac{\pi\alpha}{2}\Big)\Big\} \tag{4.6}$$

with $0 < \alpha \leq 2$, $-1 \leq \beta \leq 1$, and $C > 0$. The parameter α determines the asymptotic decay of $P_L(x) \sim x^{-1-\alpha}$ at $x \gg C^{1/\alpha}$, and β determines its skewness. The probability density function (PDF) belonging to Eq. (4.6) has the 'stability' property that the sum of independent identically Lévy distributed variates is Lévy distributed as well. While rapidly converging series expansions [26] of $P_L(x)$ are available for $1 < \alpha \leq 2$ or large arguments x, the evaluation of $P_L(x)$ at small arguments and $\alpha < 1$ is more involved. We use here a strategy where $P_L(x)$ is obtained from direct computation of the Fourier inverse $P_L(x) = (2\pi)^{-1} \int e^{-isx} \phi_L(s)\, ds$, with the integrand split into regimes of different approximations. At small $|s|$, both the exponential in $\phi_L(s)$ (Eq. 4.6) and the Fourier factor e^{-isx} are expanded; at larger $|s|$, $\phi_L(s)$ is piecewise expanded while e^{-isx} is retained. In both cases, the s-integration can be done analytically, and the pieces are summed numerically. The resulting (Poisson) maximum-likelihood estimates of the parameters (α, β, C) are $\alpha_0 = 0.75$, $\beta_0 = -0.26$, $C_0 = 0.035$. The predicted frequencies are shown in Fig. 2 (gray line), and inlets represent sections of constant likelihood in (α, β, C)-space, with the 99% confidence region enclosed by boldface line. The finding $\alpha_0 < 2$ agrees with the presence of large momentum jumps.

In a next step we have investigated the waiting times $\Delta t_n = t_{n+1} - t_n$ between subsequent encounters with the dissipation regions. Simple ballistic transport between randomly positioned dissipation regions would predict a PDF of the form $f(|v_n|\Delta t_n)$ with v_n the particle velocity and $f(x)$ the PDF of distances between (magnetically connected) dissipation regions. (There is no Jacobian $d(\Delta t)/dx$ since we are dealing with discrete events.) This is in fact the case. Figure 3 (left) shows a scatter plot of the actual velocity v_n versus waiting time Δt_n. Gray crosses represent all simulated encounters with dissipation regions, including all particles and all simulated times. There is a clear trend for Δt_n to scale with v_n^{-1}, and the black solid line represents a best fit of the form $\Delta t_n = L/v_n$ with $L = 9 \cdot 20^3\, c/\Omega$. When a similar scatter plot of velocity versus energy gain is created (Fig. 3 right), then no clear correlation is seen: the energy gain is apparently independent of energy. In this sense the dissipation regions 'erase' the memory of the incoming particles. Returning to the waiting times, we may ask for the shape of the function $f(|v_n|\Delta t_n)$. This can be determined from a histogram of the quantity $|v_n|\Delta t_n$ (Fig. 4, solid line). The decay is roughly exponential (dashed: best-fit), although the limited statistics does certainly not allow to exclude other forms.

5 Summary and Discussion

We have performed exact orbit integrations of electrons in a Gauss field proxy of MHD turbulence with super-Dreicer electric fields localized in dissipation regions. It was found that the electrons remain adiabatic (during the duration of the simulation) if no dissipation regions are encountered, and can become accelerated if such are met. The resulting acceleration is intermittent and is not well described by a diffusion process, even if the underlying electromagnetic fields are Gaussian. On time scales which are large compared to the gyro time, the kinetic momentum performs a Lévy flight rather than a classical Brownian motion. The net momentum increments in the dissipation regions are independent of the in-going momentum, and have heavy tails which may be approximated by a stable law of index 0.75. The waiting times between subsequent encounters with dissipation regions are approximately exponentially distributed, $P(\Delta t) \sim e^{-v\Delta t/L}$, indicating that the dissipation regions are randomly placed along the magnetic field lines. This one-dimensional Poisson behaviour is most likely caused by the Gauss field approximation, and the waiting times in true MHD turbulence are expected to behave differently. An ongoing study is devoted to these questions, and results will be reported elsewhere. In summary, our numerical results suggest that the acceleration process may be modeled by a continuous-time random walk with finite or infinite mean waiting time, and infinite variance of the momentum increments. Such models can be described in terms of fractional versions [27,28] of the Fokker-Planck equation.

References

1. Miller, J., Cargill, P. J., Emslie, A. G., Holman, G. D. Dennis, B. R., LaRosa, T. N., Winglee, R. M., Benka, S. G., Tsuneta, S., 1997, *J. Geophys. Res*, 102, 14.631
2. Karimabadi, Menyuk, C.R., Sprangle, P., & Vlahos, L. 1987, *Astrophys. J.*, 316, 462
3. Miller J., & R. Ramaty, 1987, *Solar Phys.*, 113, 195
4. Miller J., LaRosa, T.N., & Melrose, R. L., 1996, *Astrophys. J.*, 461, 445
5. Miller, J., J.A., 1997, *Astrophys. J.*, 491, 939
6. Biskamp, D., & Müller, W.C., 2000, *Phys. Plasmas*, 7, 4889
7. Treumann, R., and Baumjohann, W., 1997, *Basic Space Plasma Physics*, Imperial College Press
8. Papadopoulos, K. 1997, in: *Dynamics of the Magnetosphere*, ed. Akasofu & Reidel, Dordrecht
9. Parker, E. N., 1993, *Astrophys. J.*, 414, 389
10. Dreicer, H. 1960, *Phys. Rev.*, 117, 329
11. Parker, E. N., 1983, *Astrophys. J.*, 264, 635
12. Matthaeus, W.H. & Lamkin, S.L., 1986, *Phys. Fluids*, 29, 2513
13. Ambrosiano, J., Mattheus, W.H., Goldstein, M.L., and Plante, D., 1988, *J. Geophys. Res.*, 93, 14.383
14. Anastasiadis, A., Vlahos, L., & Georgoulis, M. K., 1997 *Astrophys. J.*, 489, 367
15. Dimitruk, P., Matthaeus, W.H., Seenu, N., & Brown, M.R. 2003, *Astrophys. J. Lett.*, 597, L81
16. Moriyashu, S., Kudoh, T., Yokoyama, T., Shibata, K., 2004, *Astrophys. J*, 601, L107
17. Arzner, K., and Vlahos, L., 2004, *Astrophys. J. Lett.*, 605, L69
18. Karimabadi, H. N. Omidi, & C.R. Menyuk 1990, *Phys. Fluids B*, 2, 606
19. Schlickeiser R. 2002, *Cosmic Ray Astrophysics*, Berlin, Springer
20. Gardiner C. W. 1985, *Handbook of stochastic methods*, Springer.
21. Littlejohn, R. G., 1982, *J. Math. Phys.*, 23(5), 742
22. Littlejohn, R. G., 1983, *J. Plasma Physics*, 29, 111
23. Adler, R. J., 1981, *The Geometry of Random Fields*, John Wiley & Sons
24. Büchner, J., and Zelenyi, L. M, 1989, *J. Geophys. Res.*, 94, 11.821
25. Cargill, P. J., 2001, in: Encyclopedia of Astronomy and Astrophysics – Solar flares: Particle Acceleration Mechanisms, Nature Publ. Group, IOPP, Bristol, UK
26. Lukacs, E., 1960, *Characteristic Functions*, Charles Griffin, London
27. Uchaikin, V. V., 2000 *Int. J. of Theoretical Physics*, 39, no 8, 2087
28. Meerschaert M. M. and Benson, D. A, 2002, Phys. Rev. E, 66, 060102(R)

A Simulation Model for Forest Fires

Gino Bella[1], Salvatore Filippone[1], Alessandro De Maio[2], and Mario Testa[2]

[1] Università di Roma "Tor Vergata", Rome, Italy
{bella,salvatore.filippone}@uniroma2.it
[2] Numidia s.r.l., Rome, Italy

Abstract. The control of forest fires is a very important problem for many countries around the world. Proper containment and risk management depend on the availability of reliable forecasts of the flame front propagation under the prevailing wind conditions, taking into account the terrain features and other environmental variables.
In this paper we discuss our initial development of the central part of an integrated system, namely a tool to simulate the advancement of the flame front. We discuss the pyrolisis model, the wood characteristics taken into account, and the modeling of heat exchange phenomena; this modeling tool is derived from the well known fluid dynamics code Kiva, originally developed for engine design applications.
Finally, we give an overview of the future directions of development for this activity, with special attention to the models of combustion employed.

1 Introduction

Wood combustion as encountered in a forest fire is a very complex phenomenon with many interactions.

One of the main tools employed so far in the study of fires is the use of a special purpose wind tunnel with a combustive bed, i.e. a layer of material reproducing the combustion characteristics of the wood mix in the area under consideration.

The simulation tool that has been developed has now reached a development status sufficient to replicate most results from a wind tunnel experiment; this is of course the first step towards the ability to simulate a fire in a given terrain scenario. The simulation engine is coupled with pre and post processing tools, to provide a complete visualization of the simulation scenario.

Our simulation tool is based on the Kiva code [1]; KIVA is a code capable of simulating chemically reacting fluid flows, originally developed in the context of engine simulations.

2 The Wind Tunnel Fire Bed Simulation

The fire bed is a layer of combustive material with a parallelepiped shape. It is considered to be composed of spherical particles; the material is characterized by the percentage of cellulose, water and inert and makes up a permeable bed, with a user-specified ratio between solid and gas components.

The base environment is the KIVA-3 code, originally developed at Los Alamos for the simulation of internal combustion engines and related gas-fuel spray interactions. Its mathematical governing equations and Numerical method are described below.

To simulate wood combustion, we had to expand the base code by integrating models for heat exchange between gas and wood particles, and for pyrolisis.

The current simulation model is already adequate to represent undergrowth fire; in the next development cycle we plan to introduce the model for massive solid wood entities, i.e. trees.

2.1 The Mathematical Governing Equations

The mathematical model of KIVA-3 is the complete system of general *time dependent Navier-Stokes equations*, coupled with chemical kinetic and spray droplet dynamic models. The fluid motion equations are:

– Species continuity:

$$\frac{\partial \rho_m}{\partial t} + \nabla \cdot (\rho_m \mathbf{u}) = \nabla \cdot [\rho D \nabla (\frac{\rho_m}{\rho})] + \dot{\rho}_m^c + \dot{\rho}_m^s$$

where ρ_m is the mass density of species m, ρ is the total mass density, D is the diffusion coefficient, $\mathbf{u}$ is the fluid velocity, $\dot{\rho}_m^c$ is the mass source term due to chemistry, $\dot{\rho}_m^s$ is an additional mass source term. By summing the previous equation over all species we obtain the total fluid density equation:

– Total mass conservation:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = \dot{\rho}^s$$

– Momentum conservation:

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\, \mathbf{u}) =$$
$$-\frac{1}{\alpha^2} \nabla p - A_0 \nabla (\frac{2}{3} \rho k) + \nabla \cdot \overline{\sigma} + \mathbf{F}^s + \rho \mathbf{g}$$

where α is a dimensionless quantity used for the numerical PGS method [1], $\overline{\sigma}$ is the viscous stress tensor, $\mathbf{F}^s$ is the rate of momentum gain per unit volume due to interactions between gas and other phases and $\mathbf{g}$ is the constant specific body force. The quantity A_0 is zero in laminar calculations and unity when turbulence is considered, k is the turbulent kinetic energy and ϵ its dissipation rate.

– Internal energy conservation:

$$\frac{\partial (\rho I)}{\partial t} + \nabla \cdot (\rho I \mathbf{u}) =$$
$$-p \nabla \cdot \mathbf{u} + (1 - A_0) \overline{\sigma} : \nabla \mathbf{u} - \nabla \cdot \mathbf{J} + A_0 \rho \epsilon + \dot{Q}^c + \dot{Q}^s$$

where I is the specific internal energy, the symbol : indicates the matrix product, $\mathbf{J}$ is the heat flux vector, $\dot{Q}^c$ is the source terms due to chemical heat release $\dot{Q}^s$ is an additional source term.

Furthermore, two additional transport equations are considered. These are the standard $k - \epsilon$ equations for the turbulence including terms due to interaction with different phases [1].

2.2 Numerical Method

The numerical method employed in KIVA-3 is based on a variable step *implicit Euler* temporal finite difference scheme, where the timesteps are chosen using accuracy criteria. Each time step defines a cycle divided in three phases, corresponding to a physical splitting approach.

In the first phase the chemical kinetic equations are solved, providing most of the source terms; the other two phases are devoted to the solution of fluid motion equations. The spatial discretization of the equations is based on a *finite volume method*, called the *Arbitrary Lagrangian-Eulerian method*, using a mesh in which positions of the vertices of the cells may be arbitrarily specified functions of time. This approach allows a mixed Lagrangian-Eulerian flow description. In the KIVA Lagrangian phase, the vertices of the cells move with the fluid velocity and there is no convection across cell boundaries. In this phase, the diffusion terms and the terms associated with pressure wave propagation are implicitly solved by a modified version of the SIMPLE (Semi Implicit Method for Pressure-Linked Equations) algorithm [2]. This algorithm, well known in the CFD community, is an iterative procedure to compute velocity, temperature and pressure fields. Upon convergence on pressure values, implicit solution of the diffusion terms in the turbulence equations is approached.

After the SIMPLE solver has converged, we have an explicit convection phase, in which the grid is adapted to the new physical condition if needed. In our fire-bed simulation we have an advantage with respect to the standard code applications, in that there is no rezoning, i.e. the computational mesh does not change during the simulation. However we also have to account for the heat exchange by radiation in the first phase.

2.3 Heat Exchange Models

Heat exchange between solid wood particles and sorrounding gas in a fire simulation happens through conduction, convection and radiation.

The heat conduction part is modeled by a fifth order polynomial expression. The convection effect is biased through a number assigned to the Nusselt number; experimental evidence shows that a value around 2 is appropriate.

The radiation heat exchange is modeled through the use of a vision factor [4]; this in turn is computed from the solid angle under which the cell under consideration "perceives" a solid body within the cell (see Fig. 1).

To limit the computational requirements we define a cutoff distance beyond which the effect of radiation exchange is heuristically neglegible and would be only a computational cost. The source energy is computed with the formula

$$\dot{Q}(T, species) = 4\sigma \sum_i p_i a_{p_i} (T^4 - T_b^4)$$

where

- $\sigma = 5.6669 \times 10^{-8} \frac{W}{m^2 K^4}$ is the Stefan-Boltzmann constant;
- The sum is extended over all the (gaseous) chemical species produced in the combustion;

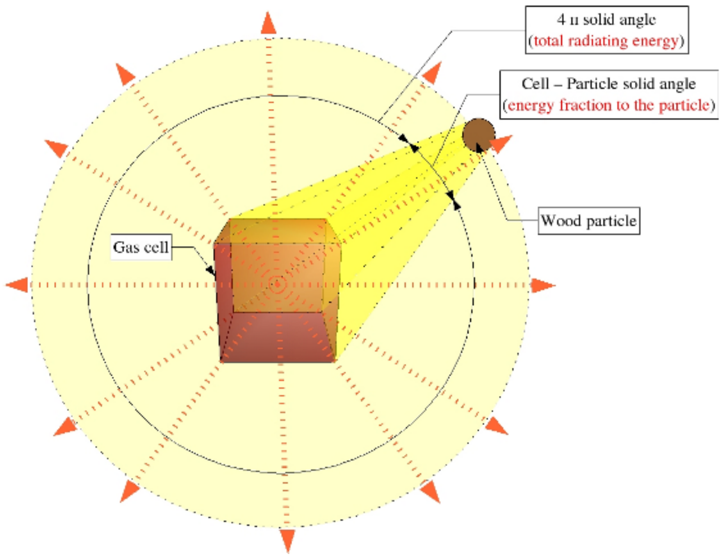

4 π solid angle
(total radiating energy)
Cell – Particle solid angle
(energy fraction to the particle)
Wood particle
Gas cell

Fig. 1. Radiation heat exchange model

- p_i is the partial pressure of the i-th species;
- a_{p_i} is the Planck absorption coefficient for the i-th species;
- T is the local flame temperature;
- T_b is the absolute temperature of the sapce surrounding the flame.

Similarly to the cutoff distance, we introduce a cutoff temperature below which the radiated thermal flux is zeroed. It is also possible to introduce a correction factor F_a for the attenuation of the radiation efficiency; the attenuation is modeled with a flat region, followed by a linearly decreasing region.

2.4 Pyrolisis Model

Due to thermal exchange, the particle temperature increases and the water in it vaporizes, providing a source term in mass, momentum and energy equations. After vaporisation is complete the pyrolisis process starts. For our purposes, pyrolisis corresponds to the thermal decomposition of organic molecules taking place without oxidizing agents following this chemical equation [5,6,7]:

$$C_6H_{10}O_5 -> 3.74C + 2.65H_2O + 1.17CO_2 + 1.08CH_4$$

The pyrolisis rate, as well as that of the oxidation process, is modeled with an Ahrrenius type kynetic equation for the mass gradient

$$\frac{dm}{dt} = A_m e^{(-\frac{E_a}{RT})}$$

where A_m is the pyrolisis reactivity and E_a is the activation energy necessary to trigger the process. A similar model controls the char creation process; for further details see [8]. All the mass, momentum and energy contributions of this processes are considered in the corresponding governing equations.

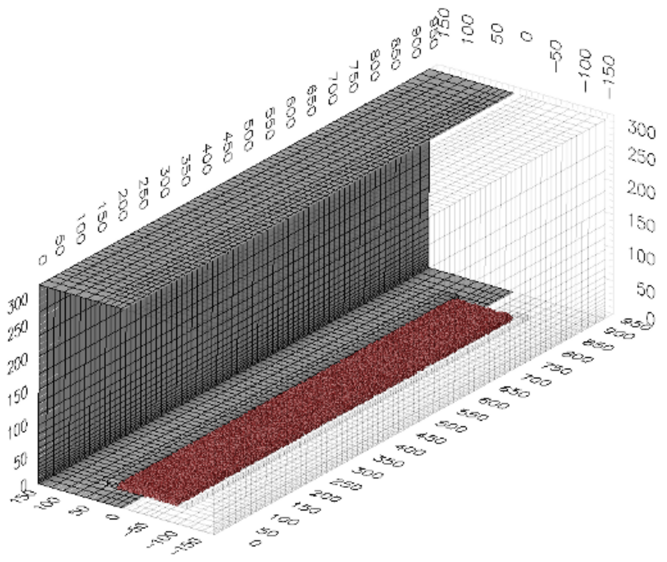


Fig. 2. Fire-bed discretization mesh geometry, measures in cm

3 Computational Experiments

The test case we present has been built to reproduce the experimental setting described in [9]. The corresponding computational mesh is shown in Fig. 2; it has 27489 total cells, containing 20000 combustive particles. The combustive material bed composed of "Pines Ponderosa Needles" is 0.076 m thick, with a packing ratio of 0.063 and moisture (ratio of water mass to dry mass) of 0.264. The air speed is 2.68 m/s, and the experimental flame front speed is 0.008 m/s.

The simulation results are shown in Fig. 3 and 4, where we can see the flame front position at two different simulated times; the flame front is identified with the isosurface of burnt gases at a temperature of 500 K.

In Fig. 5 we show the flame front propagation speed as a function of simulated time; the simulated propagation speed agrees nicely with the experimental result of 0.008 m/s.

The computational requirements are quite heavy; the simulation covering about 800 s shown above has taken over 100 hours on a standard workstation equipped with an Intel Pentium IV processor at 2.8 GHz, with 1 GB of main memory. The introduction of the advanced solvers used in [3] has already given some benefits as detailed in Fig. 6 where we show the time needed to simulate one second of the physical phenomenon for the old

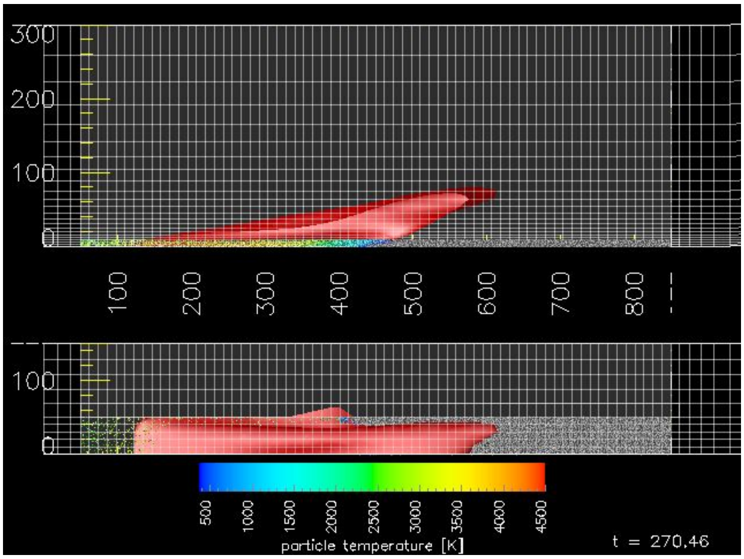


Fig. 3. Flame front visualization at 270 seconds

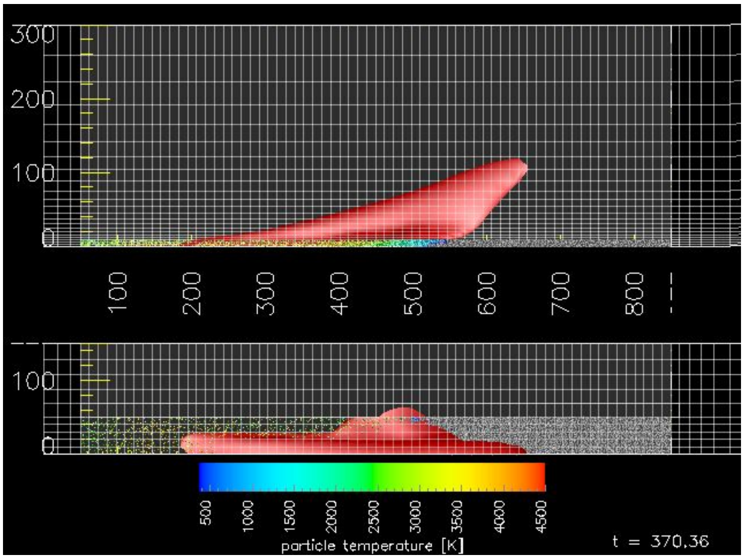


Fig. 4. Flame front visualization at 370 seconds

and new version of the code, over the first 30 time steps. As we can see, the new solvers are more efficient as soon as the combustion starts; since the overall simulation takes more than 20000 time steps, dominated by the combustion, the algorithmic change alone gives almost a factor of 2 speedup. Moreover the solvers are suitable to run in parallel;

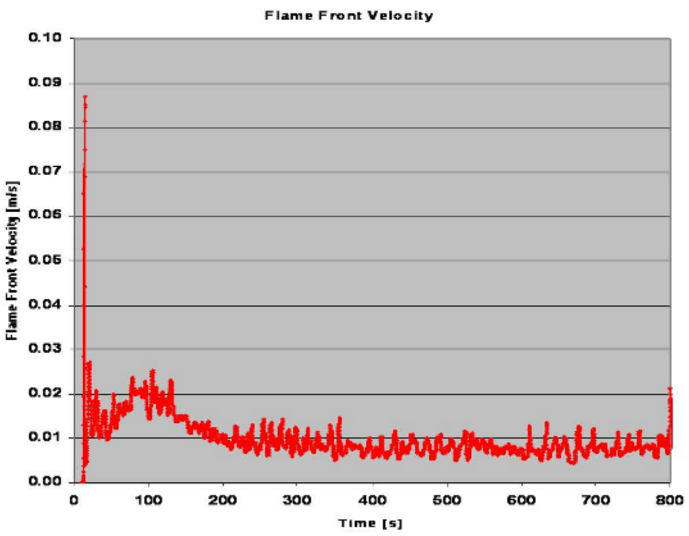


Fig. 5. Flame front velocity (calculated from the burning rate)

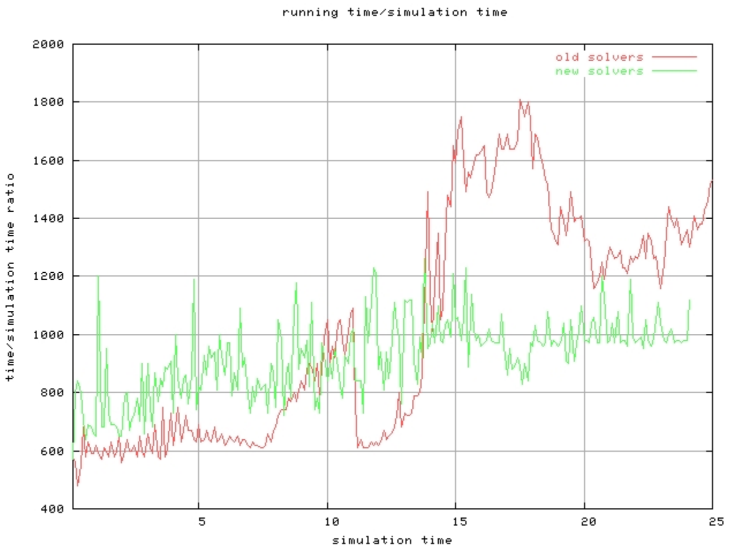


Fig. 6. Simulation runtime

we are currently actively working on the parallelization of the heat exchange model, to obtain a reasonable run time for the physical cases of interest.

4 Development Directions

The first development direction is related to the implementation of the computational model engine; we have already started the parallelization of the code by merging the solver modules developed for the engine application [3].

From the modeling point of view the most important issues to be tackled are:

– Modeling of trees
– Realistic terrain shapes

These are under consideration, and we plan to address them in medium term developments.

The long term vision for this project is to embed this engine into an integrated system comprising real-time satellite data feeds and interfaces with GIS databases to provide a flexibile tool for risk management; this is a very challenging undertaking that will require substantial computational resources as well as a significant amount of code development.

References

1. A.A. Amsden, *KIVA 3: A KIVA Program with Block Structured Mesh for Complex Geometries*, Los Alamos National Lab., Tech. Rep. LA-12503-MS, 1993.
2. S.V. Patankar, *Numerical Heat Transfer and Fluid Flow*, Hemisphere Publ. Corp., 1980.
3. S. Filippone, P. D'Ambra, and M. Colajanni. *Using a Parallel Library of Sparse Linear Algebra in a Fluid Dynamics Applications Code on Linux Clusters.* In G. Joubert, A. Murli, F. Peters, M. Vanneschi eds., *Parallel Computing - Advances & Current Issues*, Imperial College Press Pub., pp. 441–448, 2002.
4. W. L. Grosshandler, *RADCAL: A narrow-Band Model for Radiation Calculations in a Combustion Environment*, NIST techical note 1402, 1993.
5. F. Shafizadeh, *The Chemistry of pyrolysis and combustion.* In: R.M. Rowell (ed.) 1984 The Chemistry of Solid Wood, Advances in Chemistry Series 207. American Chemical Society, Washington, DC, pp 489–530.
6. A. J. Stamm, *Thermal degradation of wood and cellulose.* Presented at the Symp. On Degratadion of Cellulose and Cellulose Derivatives, 127th National Meeting of the American Chemical Society, Cincinnati Ohio, 1955.
7. F. Shafizadeh, *Chemistry of Pyrolysis and Combustion of Wood*, in Proceedings of 1981 International Conference on Residential Solid Fuels - Environmental Impacts and Solutions, Cooper JA, Malek Deds, Oregon Graduate Center, Beverton: pp 746–771.
8. J. Hong, *Modelling of char oxidation as a function of pressure using an intrinsic Langmuir rate equation*, Tech. Rep. Brigham Young University, 2000.
9. W.R. Catchpole, E.A. Catchpole, B.W. Butler, R.C. Rothermel, G.A. Morris, D.J. Latham, *Rate of Spread of Free-Burning Fires in Woody Fuels in a Wind Tunnel* USDA Forest Service, Rocky Mountain Research Station, Intermountain Fire Sciences Laboratory, Missoula.

MHD Modeling of the Interaction Between the Solar Wind and Solar System Objects

Andreas Ekenbäck and Mats Holmström

Swedish Institute of Space Physics (IRF)
P.O. Box 812
98134 Kiruna, Sweden
{andreas.ekenback,mats.holmstrom}@irf.se

Abstract. Magnetohydrodynamic (MHD) models of plasma can be used to model many phenomena in the solar system. In this work we investigate the use of a general MHD solver - the Flash code - for the simulation of the interaction between the solar wind and solar system objects. As a test case we simulate the three-dimensional solar wind interaction with a simplified model of a comet and describe the modifications of the code. The simulation results are found to be consistent with previously published ones. We investigate the performance of the code by varying the number of processors and the number of grid cells. The code is found to scale well. Finally we outline how to simulate the solar wind interaction with other objects using the Flash code.

1 Introduction

The solar wind is a plasma consisting mostly of protons and electrons that are ejected from the Sun's corona, and streams radially outward through the solar system at speeds of several hundred kilometers per second. Embedded into the solar wind is also the interplanetary magnetic field. When this highly supersonic solar wind meets with solar system objects, such as planets, comets and asteroids, an interaction region is formed. The type of the interaction depends on the object. Around planets with a strong internal magnetic field, such as Earth, a cavity is formed, a magnetosphere, shielding the upper atmosphere from direct interaction with the solar wind. This shielding is missing at planets without a strong intrinsic field, e.g., at Mars. Then currents in the planet's ionosphere form an obstacle that diverts the solar wind flow. Objects without any significant atmosphere, such as the Moon, are basically just physical obstacles to the flow, and a wake is formed behind the object. Comets have a very small nucleus compared to the size of the surrounding cloud of ions and neutrals and their interaction with the solar wind is governed by photoionization of neutrals and charge-exchange between ions and neutrals.

In space physics, the numerical modeling of the interaction between the solar wind and solar system objects is an important tool in understanding the physics of the interaction region. The results of simulations is also important as inputs to further modeling, e.g., to predict loss of planetary atmospheres due to the interaction.

Although self consistent particle and kinetic models best capture the actual physics, the computational cost at present limits the number of particles and the refinement of grids. Since the computational cost of using a fluid model is less, more refined grids can be used. Thus, the modeling of the interaction between the solar wind and solar system objects is often done by using magnetohydrodynamic (MHD) fluid models.

In this work we investigate the possibility of using an existing open source application - the Flash code [1] developed at University of Chicago - to make MHD simulations of the interaction between the solar wind and a non-magnetized object. We model a solar wind and a comet - in the form of a photoionization source - and outline how the simulation model could be generalized to simulate other objects in the solar system.

2 Magnetohydrodynamics

Over small spatial volumes, the average properties of a plasma can be described using the basic conservation laws for a fluid [2]. The plasma is however conducting, so the effects of electric and magnetic fields and currents are added to ordinary hydrodynamics. In the presence of a magnetic field, the added conductivity of a fluid will cause a different behavior due to the interaction between the magnetic field and the motion; electric currents induced in the fluid (as a result of its motion) modify the field, and at the same time the flow of the currents in the field causes mechanical forces which modify the motion.

2.1 MHD Equations

The MHD model used for most simulations is valid under a number of assumptions, listed in full in [3], the fundamental ones being that the time scale of interest must be long compared to microscopic particle motions and that the spatial scale of interest must be large compared to the Debye length and the thermal gyroradius [2]. MHD models have nevertheless proved adequate to describe several physical processes in the solar system [4,5,3], including the solar wind-comet interaction [6,7].

In what follows we use the notation:

$\mathbf{r}$	is position
ρ	is mass density
$\mathbf{v}$	is the velocity of the fluid
$\mathbf{B}$	is the magnetic field
μ_0	is permeability in vacuum $= 4\pi \times 10^{-7}$ Vs/Am
p	is the plasma pressure
ε	is the internal energy
$\mathbf{g}$	is external forces such as gravity
T	is the temperature of the fluid
t	is elapsed time

It is common to use the MHD equations in their dimensionless, or normalized, form. We choose to use a reference velocity v_0, a reference density ρ_0 and a reference length R_0. These three factors are then used to scale all quantities in the MHD equations. Letting an overbar indicate the variables in SI units we set

$$\mathbf{r} = \frac{\bar{\mathbf{r}}}{R_0} \qquad \rho = \frac{\bar{\rho}}{\rho_0} \qquad \mathbf{v} = \frac{\bar{\mathbf{v}}}{v_0}$$

which for the other interesting parameters leads to

$$t = \frac{v_0}{R_0}\,\bar{t} \qquad \mathbf{B} = \frac{\bar{\mathbf{B}}}{\sqrt{\mu_0 \rho_0 v_0^2}} \qquad T = \frac{\bar{T}}{v_0^2} \qquad p = \frac{\bar{p}}{\rho_0 v_0^2}$$

Approximating the plasma as collisionless and without resistivity we can use the equations of ideal MHD [8,2]. Using the ideal limit of the one-fluid MHD equations in conservative form we will solve in Flash

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{2.1}$$

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot (\rho \mathbf{v}\mathbf{v} - \mathbf{B}\mathbf{B}) = -\nabla \left(p + \frac{B^2}{2} \right) + \rho \mathbf{g} \tag{2.2}$$

$$\frac{\partial \rho E}{\partial t} + \nabla \cdot \left[\mathbf{v} \left(\rho E + p + \frac{B^2}{2} \right) - \mathbf{B}(\mathbf{v} \cdot \mathbf{B}) \right] = \rho \mathbf{g} \cdot \mathbf{v} \tag{2.3}$$

$$\frac{\partial \mathbf{B}}{\partial t} + \nabla \cdot (\mathbf{v}\mathbf{B} - \mathbf{B}\mathbf{v}) = 0 \tag{2.4}$$

where

$$E = \frac{1}{2}v^2 + \varepsilon + \frac{1}{2}\frac{B^2}{\rho}$$

The MHD system is in the Flash code complemented by the equation of state, relating the pressure to internal energy or temperature:

$$\bar{p} = (\gamma - 1)\,\bar{\rho}\,\bar{\varepsilon} \qquad \bar{p} = \frac{N_a k_B}{M}\,\bar{\rho}\bar{T}$$

where

γ	is the adiabatic index
N_a	is the Avogadro number = 6.022×10^{23}
M	is the mean atomic mass [kg]
k_B	is the Boltzmann constant = 1.38×10^{-23} J/K

3 Use of the Open Source Flash Code

The Flash code is a modular, adaptive, parallel simulation code capable of handling general compressible flow problems in astrophysical environments. The Flash code is written mainly in Fortran90 and uses the Message-Passing Interface (MPI) library [9] for inter-processor communication and portability. It further uses a customized version of the PARAMESH library [10,11] to implement a block-structured adaptive cartesian grid, increasing resolution where it is needed.

By using the MHD module of the Flash code, we can benefit from all the testing, development and performance tuning that has been done. Flash is also distributed with an IDL-based visualization tool that we found very useful. What we add is the specific initial conditions, boundary conditions and source terms for the problem at hand. The various examples provided with Flash facilitates the task of setting up a simulation model.

We here explain how to add a new problem to Flash by using the solar wind-comet interaction as an example. Where needed we also indicate how to model other possible situations.

3.1 Creating a Problem Directory

The Flash code was written with the intention of users being able to easily add their specific problems. The code is hence well prepared for adding a new problem; in the source code there is a directory `$FLASHHOME$/setups` where we create a directory `comet_mhd` for our problem. There are two files that must be included in a problem setup directory:

`Config` where it is specified which of the modules in Flash that should be used. It also lists (and sets default values to) required runtime parameters. In case we specify a module in our `Config` file with several submodules, there are default ones which are used unless we also specify the submodule.

`init_block.F90` to initialize data in each of the blocks of the mesh. We give the initial values in terms of ordinary geometrical coordinates and Flash takes care of the mapping to blocks. The code first initializes the blocks at the lowest level of refinement and then where needed refines the mesh and initializes the newly created blocks. If necessary this refinement procedure is repeated until we reach a highest level of refinement.

3.2 Boundary Conditions

To specify the boundary conditions it is possible to use some of the standard ones ("outflow","periodic" or "reflecting") by a line for each boundary in the runtime parameter file. By stating a boundary as "user" Flash will use a file `user_bnd.F90` in the problem directory to set specified variable values there. Our solar wind inflow is modeled by setting velocity, density, magnetic field and temperature at the left x-boundary and setting the boundary condition at the right x-boundary as "outflow". The simulation is periodic in the y- and z-directions.

3.3 Alternation to Standard Algorithms

In order to add the comet source terms for the MHD equations as specified in Section 4 it was necessary to make alterations in the main file for MHD calculations. This main file is `mhd_sweep.F90` and it was copied to our problem directory. Alternations are then made in our `mhd_sweep.F90` and this one will be used instead of the standard one. Calls to a created file `comet_source_mhd.F90`, that do additions to the rand hand side of the equations (2.1) and (2.2), were inserted in `mhd_sweep.F90`. To see that our non-standard file `comet_source_mhd.F90` is compiled into the final executable it is necessary to have a `Makefile` in our problem directory stating the target `comet_source_mhd.o` and its dependencies.

3.4 Runtime Parameter File

The final file needed to run our comet simulation is the runtime parameter file `flash.par`. It should be placed in the directory from where Flash is run. Besides assigning values to parameters and specifying boundary conditions, it controls the output. Checkfiles are dumped at intervals specified in `flash.par` and can be used to restart simulations from. We can also control when smaller files, to be used with the enclosed visualization tool, containing only a desired selection of the variables are dumped.

4 A Test Case: The Solar Wind-Comet Interaction

To investigate the use of the Flash code for simulating the interaction between solar system objects and the solar wind we choose a comet as a test problem. We model the comet using a single fluid MHD. The setup follows Ogino et al. [6] where the comet is modeled as a spherically symmetric source of ions. Although much more refined comet models have been used later, e.g. by Gombosi et al. [7], we use this simpler model since we in this work primarily are interested in the performance and correctness of the code and our modifications.

Because of the small mass (and the resulting small gravitational forces) of the cometary nucleus, a large amount of gas evaporates from the nucleus. The gas extends into the solar wind and is ionized by charge-exchange processes and by photoionization caused by solar ultraviolet radiation. In the MHD equations the interaction process is, following Ogino et al. [6] (originally by Schmidt et al. [12]), modeled as a plasma production centered at the cometary nucleus. We set the cometary plasma production rate

$$\bar{A}(\bar{r}) = \frac{\bar{Q}_p\, e^{-\bar{r}/\bar{\lambda}_c}}{4\pi\bar{\lambda}_c\bar{r}^2} \tag{4.5}$$

where

$\bar{Q}_p$	is the mass production rate of ions = 2.656×10^4 s^{-1}kg
$\bar{r}$	is the distance to the center of the comet [m]
$\bar{\lambda}_c$	is the ionization distance = 3.03×10^5 m, that controls the width of the source

for which we make the following normalizations (in order to make $\bar{A}$ dimensionless):

$$Q_p = \frac{\bar{Q}_p}{\rho_0 v_0 R_0^2} \qquad r = \frac{\bar{r}}{R_0} \qquad \lambda_c = \frac{\bar{\lambda}_c}{R_0}$$

We modify the continuity equation (2.1) to

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = A(r) \tag{4.6}$$

The cometary ions will also cause a modification to the momentum equation (2.1) so that we have

$$\frac{\partial\rho\mathbf{v}}{\partial t} + \nabla\cdot(\rho\mathbf{v}\mathbf{v} - \mathbf{B}\mathbf{B}) = -\nabla\left(p + \frac{B^2}{2}\right) + \rho\mathbf{g} - A(r)\mathbf{v} \tag{4.7}$$

in the same region.

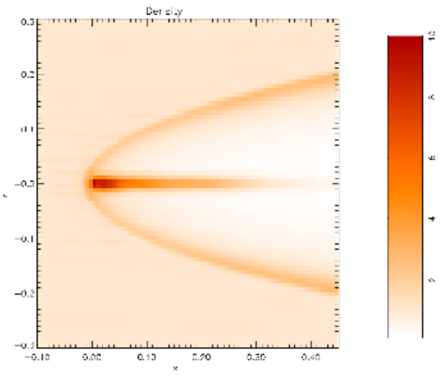


Fig. 1. Density at steadystate for our 3D simulation of a comet with parameter settings n_{sw} = 15 cm^{-3}, $\mathbf{v}_{sw} = v_x$ = 500 km/s, $\mathbf{B}_{sw} = B_z$ = 6 nT and source terms according to (4.5)-(4.7). This steady state occured $\approx$ 1 second after we add the comet as a source term, and the figure shows the density distribution 1.4 seconds after the encounter. We here show the x-z plane and thus have the solar wind coming in from the left and its magnetic field in the upward direction. The minimal refinement level was set to 2 and the maximal to 5, resulting in a peak of 1,024,000 in the number of cells

We use the same normalization as in [6], using the earth radius $R_E = 6.37 \times 10^6$ m as R_0. Other quantities are scaled with typical solar wind values: $\rho_0 = 2.51 \times 10^{-20}$ m^{-3}kg (corresponding to a number density of the solar wind n_{sw} of 15×10^6 m^{-3}) and v_0 = 500 km/s. The magnetic field of the solar wind is set so that $\bar{\mathbf{B}} = \bar{B}_z$ = 6 nT and we use a solar wind temperature $\bar{T}_{sw} = 2 \times 10^5$ K. We use an adiabatic index γ = 5/3 and include no external forces ($\mathbf{g} = 0$).

5 Results for the Test Case

We here present results for the test case of the solar wind interaction with a comet, as described in Section 4. We first present the simulation results in the physical context and compare them with expected results and previously done simulations. Thereafter we present how the code scales with problem size and when we increase the number of used processors.

From a physical point of view the simulations give result in agreement with Ogino et al. [6]. Figure 1 shows the result of a typical simulation in 3D. The characteristic weak bow shock is seen in all simulations. As seen in Figure 2, the solar wind is decelerated by the cometary ions and the magnetic field is draped and obtains peak values near the comet nucleus.

The code seems to scale well for our test problem. The execution time grows almost linearly with the problem size as seen in Figure 3. There is also profit to be made by using more processors for a larger problem as seen to the right in Figure 3 - the deviation from the linear speedup appears later when we increase the problem size. As expected the speedup is sub-linear as we increase the number of processors due to increased communication time.

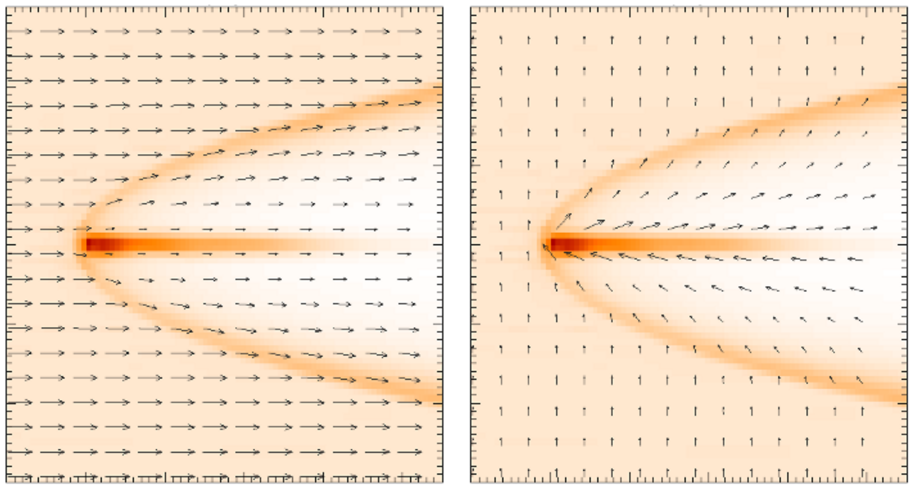


Fig. 2. Velocity (left) and magnetic (right) field vectors for our 3D simulation of a comet for comparison with Ogino et al. [6]. This corresponds to their Figure 3. Parameter values, length and density scale are the same as in Figure 1

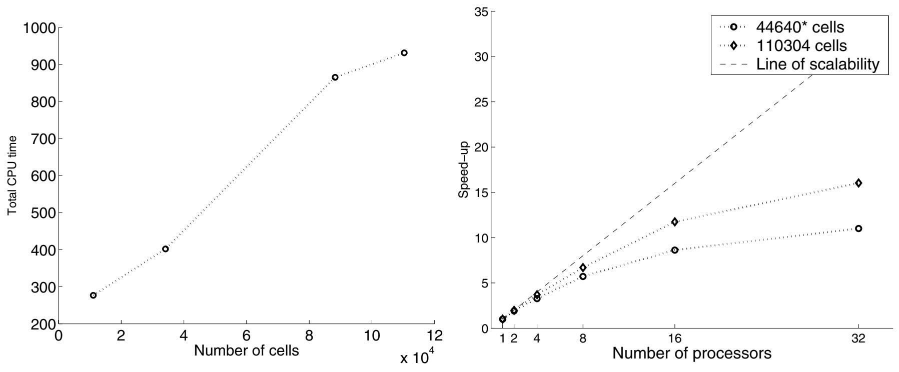


Fig. 3. Left: Execution time as function of the number of cells in the problem. To the right we show the speedup as a function of the number of processors for two problem sizes

The mesh refinement works fine for our test problem; refinements are made in critical regions and only there. The automatic refinement is thus much satisfactory. The code is also capable of handling the discontinuities of our test problem. In our simulations we start with a flowing solar wind and then insert the production of cometary ions in the middle of our simulation box. This does not cause any problem in spite of the much lower velocity of these ions. Both the automatic refinement mechanism and discontinuities are shown in Figure 4 where we show a simulation at t=0.2,0.4 and 0.6 seconds. This simulation was done in 2D so that we could have large refinements differences and still keep simulation time reasonable.

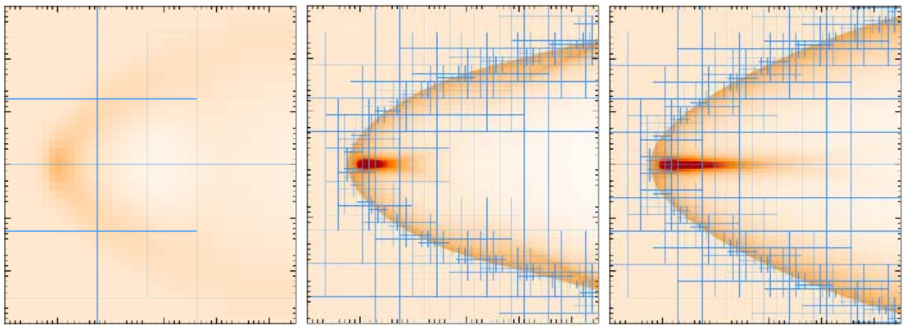


Fig. 4. The start of a simulation in 2D. At t=0 we add the source term A to the MHD equations as explained in Section 4. This shows the density at t=0.2s (left), t=0.4s (middle) and t=0.6s. Note how the mesh is refined (only) where needed. The minimal refinement level was set to 1 and the maximal to 8, resulting in a peak number of 128,000 cells

The simulations were performed at the Seth Linux Cluster at the High Performance Computing Center North (HPC2N) [13]. The cluster has AMD Athlon MP2000+ processors in 120 dual nodes with each CPU running at 1.667 GHz. The network has 667 Mbytes/s bandwidth and an application level latency of 1.46 μs. The network connects the nodes in a 3-dimensional torus organized as a 4x5x6 grid. Each node is also equipped with fast Ethernet. For all our simulations the limitation is CPU speed (and communication as we increase the number of nodes), as we do not use the full 1 GB memory of each node.

6 Conclusions and Discussion

In this work we have shown the feasibility of using a general MHD solver for the simulation of the interaction between the solar wind and solar system objects. The open source Flash code needed small changes to specify the initial conditions, boundary conditions and source terms to solve our selected model - the three-dimensional interaction between the solar wind and a spherical symmetric source region, a simplified model of a comet.

We assured the correctness of our results by comparison with previously published ones. The simulations showed that the code scales well, both as a function of problem size, and as a function of number of computing nodes.

Thus, the Flash code seems to have the possibility of beeing used as a general tool for the investigation of the interaction between solar system objects and the solar wind.

Some areas of future investigation is the possibility of solving multi fluid MHD problems (e.g., H^+ and O^+), test particle trajectory integration in the MHD solution field, and the way to handle planetary bodies.

Acknowledgments

The software used in this work was in part developed by the DOE-supported ASC /Alliance Center for Astrophysical Thermonuclear Flashes at the University of Chicago, USA.

References

1. Alliances Center for Astrophysical Thermonuclear Flashes (ASCI), http://flash.uchicago.edu
2. M.G. Kivelson and C.T. Russell (editors). *Introduction to Space Physics*, University Press, Cambridge, UK, 1995 ISBN 0 521 45104 4
3. E.R. Priest and A.W. Hood (editors). *Advances in Solar System Magnetohydrodynamics*, University Press, Cambridge, UK, 1991 ISBN 0 521 40325 1
4. H. Matsumoto and Y. Omura (editors). *Computer Space Plasma Physics*, Terra Scientific Publishing Company, Tokyo, Japan, 1993 ISBN 4 88704 111 X
5. K. Kabin, K.C. Hansen, T.I. Gombosi, M.R. Combi, T.J. Linde, D.L. DeZeeuw, C.P.T. Groth, K.G. Powell and A.F. Nagy. Global MHD simulations of space plasma environments: heliosphere, comets, magnetospheres of planets and satellites *Astrophysics and Space Science*, 274:407-421, 2000
6. T. Ogino, R.J. Walker and M. Ashour-Abdalla. A Three-Dimensional MHD Simulation of the Interaction of the Solar Wind With Comet Halley *Journal of Geophysical Research*, vol. 93, NO. A9, pages 9568-9576, September 1988
7. T.I. Gombosi, D.L. De Zeeuw, R.M. Häberli and K.G. Powell. Three-dimensional multiscale MHD model of cometary plasma environments *Journal of Geophysical Research*, vol. 101, no. A7, pages 15233-15232, July 1996
8. T. I Gombosi. *Physics of the Space Environment*, University Press, Cambridge, UK, 1998 ISBN 0 521 59264 X
9. The Message Passing Interface (MPI), http://www-unix.mcs.anl.gov/mpi/
10. Parallel Adaptive Mesh Refinement (PARAMESH), http://ct.gsfc.nasa.gov/paramesh/Users_manual/amr.html
11. P. MacNeice, K.M. Olson, C. Mobarry, R. deFainchtein and C. Packer. PARAMESH: A parallel adaptive mesh refinement community toolkit *Computer Physics Communications*, vol. 126, pages 330-354, 2000
12. H.U. Schmidt and R. Wegmann. Plasma flow and magnetic fields in comets *Comets: Gases, Ices, Grains and Plasma* edited by L.L. Wilkening. University of Arizona Press, Tuscon, USA
13. High Performance Computing Center North, Umeå, Sweden, http://www.hpc2n.umu.se

Implementing Applications with the Earth System Modeling Framework

Chris Hill[1], Cecelia DeLuca[2], V. Balaji[3], Max Suarez[4], Arlindo da Silva[4], William Sawyer[4,7], Carlos Cruz[4], Atanas Trayanov[4], Leonid Zaslavsky[4], Robert Hallberg[3], Byron Boville[2], Anthony Craig[2], Nancy Collins[2], Erik Kluzek[2], John Michalakes[2], David Neckels[2], Earl Schwab[2], Shepard Smithline[2], Jon Wolfe[2], Mark Iredell[6], Weiyu Yang[6], Robert Jacob[5], and Jay Larson[5]

[1] Massachusetts Institute of Technology
[2] National Center for Atmospheric Research
[3] Geophysical Fluid Dynamics Laboratory
[4] NASA Global Modeling and Assimilation Office
[5] Argonne National Laboratory
[6] National Center for Environmental Prediction
[7] Swiss Federal Institute of Technology
sawyer@env.ethz.ch

Abstract. The Earth System Modeling Framework (ESMF) project is developing a standard software platform for Earth system models. The standard defines a component architecture superstructure and a support infrastructure. The superstructure allows earth scientists to develop complex software models with numerous components in a coordinated fashion. The infrastructure allows models to run efficiently on high performance computers. It offers capabilities that are commonly needed in Earth Science applications, for example, support for a broad range of discrete grids, regridding functions, and a distributed grid class which represents the data decomposition. We illustrate these features through a simplified finite-volume atmospheric model, and report the parallel performance of the underlying ESMF components.

1 Introduction

Earth Science applications — for example those simulating the time evolution of the atmosphere, ocean, earth crust and mantle, or climate — traditionally have been large monolithic codes usually written by a research group at one institution. The increase in the number of environmental aspects which are simulated, along with the access to ever more powerful parallel computers, has led to a prodigious increase in the complexity of these applications.

In the last decade, researchers have acknowledged the need to share software components to overcome the software complexity. A prime example of such collaboration is a coupled ocean-atmosphere climate model, e.g. [3]. Such a model is developed by at least two groups, working for the most part independently of each other. It is key that the interfaces be clearly described, so that the separate components can be connected with a minimum of effort. Moreover, it is required that the final product runs efficiently on parallel computers. This challenge can be eased by a software framework which supports the various grids used by the components, transfers data (which are naturally distributed over the parallel machine) between grids, provides common functionality for I/O, logging, time management, and offers high-level interfaces for the interaction between components. These features are among those offered by the Earth System Modeling Framework (ESMF), a standards-based, open-source software platform for Earth Science applications.

The architecture of ESMF has been described elsewhere, e.g., [2,7]. Thus we summarize the architecture only briefly in Section 2. In Section 3, we describe the implementation of a simple atmospheric model which predates ESMF, but has been revised for ESMF compliance. The resulting application consists of a number of ESMF *components* and thus serves as a useful example for the efficacy of ESMF paradigms. Results from performance studies are presented in Section 4. Section 5 contains a brief discussion of our experiences implementing ESMF applications.

2 Architectural Overview

Software frameworks to ease development of complex applications are not new. There are several in Earth Science community alone: the GFDL Flexible Modeling System [4], the Goddard Earth Modeling System [13], the Weather Research and Forecast Model [10], the MIT Wrapper toolkit [6], the PILGRIM communications library [12] and the Model Coupling Toolkit [8]. ESMF unifies and standardizes these efforts, while offering most of the functionality of the union.

ESMF consists of a *superstructure* and an *infrastructure*. The superstructure provides a software layer encompassing user code and describes the component interconnection through input and output data streams. The infrastructure is a standard support library which developers can use to speed application development. It supplies functionality for describing physical grids and fields, managing time and calendars, logging and profiling, performing communication, regridding data, among other things. Figure 1 depicts the architecture succinctly.

Both ESMF layers provide complete support for parallel platforms. The grids used in the applications are distributed over a set of *decomposition elements* (DEs). The associated complexity of the data distribution is hidden in the superstructure, while the infrastructure has explicit routines operating on DEs.

2.1 Superstructure Layer

The superstructure layer offers a unifying context for the interconnection of components. User components which abide by the layer's standard can be inserted into other ESMF-compliant applications. Components are split into two classes: *gridded components* which perform work on a given discrete physical grid, and *coupler components* which map one state to another. The latter therefore provides the connectivity or "glue" for the former.

Central to the superstructure is the ESMF *state* class. This defines a container for component data exchanges. Thus an ESMF component accepts an *import state* and produces an *export state*. The ESMF *clock* provides consistent notions of time between

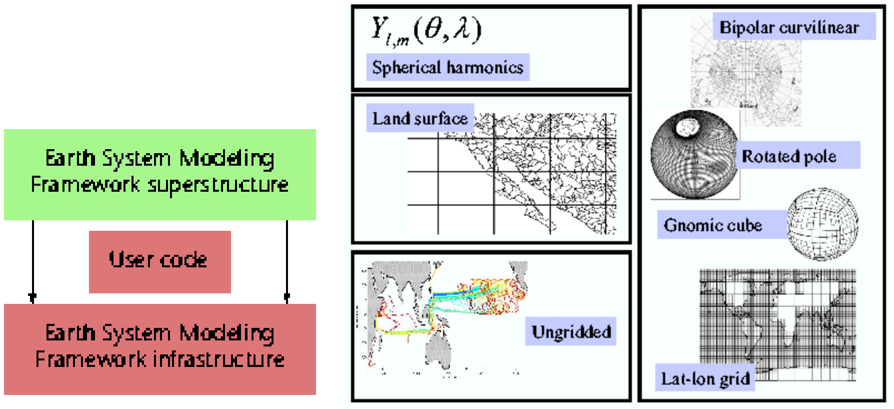

Earth System Modeling Framework superstructure
User code
Earth System Modeling Framework infrastructure
$Y_{l,m}(\theta,\lambda)$
Spherical harmonics
Land surface
Ungridded
Bipolar curvilinear
Rotated pole
Gnomic cube
Lat-lon grid

Fig. 1. The ESMF "sandwich" architecture, on left, has an upper-level (superstructure) and a lower-level (infrastructure) layer. Arrows illustrate that the superstructure-layer code (as well as the user code) can call the infrastructure layer. On right, examples of some of the horizontal grids for which ESMF provides built-in regridding support

components. Thus the import/export states and clock make up the interface for a component.

An ESMF *application* can be summarized by the following points:

1. The application is structured into a set of gridded and coupler components.
2. The underlying ESMF component (either gridded or coupler) offers the methods `Initialize`, `Run` and `Finalize`.
3. Data transferred between components are packed into ESMF import and export states.
4. Time information is packed into ESMF time management data structures.
5. ESMF superstructure functionality is used to instantiate the active gridded and coupler components.
6. Each component's `Initialize`, `Run` and `Finalize` methods are registered with the ESMF framework in the component's `SetServices` method.
7. The actual application consists only of a call to an ESMF application driver.

2.2 Infrastructure Layer

The infrastructure layer supplies functionality for tasks which are needed in a wide range of Earth Science applications. In this sense it can be considered a software library. It contains a number of classes which describe commonly used objects. For example, the import and export states mentioned in Section 2.1 consist of ESMF *fields*, which in turn consist of ESMF *arrays* and ESMF *grids* on which the data are defined. The grid is a pairing of an ESMF *physical grid* and a *distributed grid* object.

The *physical grid* describes the discretization of continuous three-dimensional space for Earth system models. There are a number of common physical grids supported, a subset of which is depicted in Figure 1. The physical grid class is extensible, allowing external groups to develop further grids for new applications.

The *distributed grid* class represents the data structure's decomposition into subdomains — typically for parallel processing. The class holds information for halo exchange of tiled decompositions in finite-difference, finite-volume and finite-element codes.

The *regrid* class allows a field F containing data on a given physical grid P and distribution grid D to be remapped to a field F', physical grid P' and distribution D'. In other words, $R_g : (F, P, D) \longrightarrow (F', P', D')$. If the $P' \neq P$, the values are interpolated, with the possibility of conserving physical quantities during the mapping. If $P' = P$, the mapping is a redistribution, for which optimized routines are available.

The *decomposition element layout* class is an abstraction for the parallel platform, which could range from a network of workstations to a distributed cluster of shared memory nodes. Parallelism is allowed with both threads and processes, both referred to abstractly as *persistent execution threads*, or PETs.

ESMF gridded and coupling components are synchronized through a common notion of time referred to as a *clock*. The *time* and *calendar* classes support a wide range of concepts of time used in Earth Science applications. For example, some research scenarios call for a year of 360 "days", while others require time to be carried in rational fractions to avoid numerical rounding and associated clock drift.

The infrastructure also defines a set of *I/O* classes for the storage and retrieval of field and grid information. These support the standard formats in the community, namely NETCDF, HDF5 and GRIB.

3 The Finite-Volume Held-Suarez Atmospheric Model

The Finite-Volume Held-Suarez (FVHS) application is a simple atmospheric model which incorporates a finite-volume dynamical core [9] to simulate atmospheric flow and idealized Held-Suarez Physics [5] which finds a rough approximation for the tendencies of physical processes influencing the atmosphere.

The coupling of FV with HS — in particular with respect to their parallel execution on supercomputers — illustrates several challenges in the development of ESMF applications, and as such serves as a case-study for the transition of existing applications to ESMF compliance.

3.1 Creating ESMF Components

The FV and HS components were already split logically into three phases of execution:

	Finite-Volume	Held-Suarez
Initialize	Read configuration Allocate internal state Read restart file Initialize variables Initialize comm. patterns	Read configuration Allocate internal state Read restart file Initialize variables
Run	Get newest input data Regrid data to internal vars. Update atmospheric state Deliver output vars.	Get newest input data Calculate tendencies Deliver output vars.
Finalize	Save internal state Deallocate variables	Save internal state Deallocate variables

The compartmentalization of the pre-ESMF code into these three areas was fortuitous for the construction of the ESMF `Initialize`, `Run` and `Finalize` methods. In the FVHS application, these are registered in the framework (Code Example 1) in the component's `SetServices` routine which is the only public access point to the component.

Code Example 1: *Registration of methods and allocation/linking of the internal state in the FV component. The* `wrap` *variable allows access to a user defined part of the component's internal state. Thus, it is possible for the developer to extend and tailor the state information for the particular application.*

```
call ESMF_GridCompSetEntryPoint ( gcFV, ESMF_SETINIT, Initialize, ESMF_SINGLEPHASE, status)
call ESMF_GridCompSetEntryPoint ( gcFV, ESMF_SETRUN, Run, ESMF_SINGLEPHASE, status)
call ESMF_GridCompSetEntryPoint ( gcFV, ESMF_SETFINAL, Finalize, ESMF_SINGLEPHASE, status)
allocate( dyn_internal_state, stat=status )
wrap%dyn_state => dyn_internal_state
call ESMF_UserCompSetInternalState ( gcFV,'FVstate',wrap,status )
call ESMF_GridCompGetInternalState ( gcFV, GENWRAP, STATUS )
```

The main program is responsible for creating the gridded components, couplers and import/export states, registering the components methods with a call to `ESMF_GridCompSetServices`, then initializing, running and finalizing the component with the corresponding ESMF calls. See Code Example 2.

Code Example 2: *FVHS initializations related to the FV component. The component is created, as well as its import and export state. Two couplers are created to connect the FV and HS components in both directions. The* `Initialize`*,* `Run` *and* `Finalize` *services (whose entry points were described previously) are registered with the framework with* `ESMF_GridCompSetServices`*. Thereafter it is possible to access these services with calls to the framework.*

```
FV   = ESMF_GridCompCreate ( 'FV', layout=loFV, mtype=ESMF_ATM, configfile=cf_file, rc=rc )
impFV = ESMF_StateCreate ( 'FV_Imports', ESMF_STATEIMPORT, compname='FV dyn', rc=rc )
expFV = ESMF_StateCreate ( 'FV_Exports', ESMF_STATEEXPORT, compname='FV dyn', rc=rc )
ccFVxHS = ESMF_CplCompCreate ( 'FV to HS', layout=loAppl, configfile='FVxHS.rc', rc=rc)
ccHSxFV = ESMF_CplCompCreate ( 'HS to FV', layout=loAppl, configfile='HSxFV.rc', rc=rc )
call ESMF_GridCompSetServices ( gcFV, FV_SetServices, rc )
 :
call ESMF_GridCompInitialize ( gcFV, impFV, expFV, clock, ESMF_SINGLEPHASE, rc )
 :
call ESMF_GridCompRun ( gcFV, impFV, expFV, clock, ESMF_SINGLEPHASE, rc )
 :
call ESMF_GridCompFinalize( gcFV, impFV, expFV, clock, rc)
```

3.2 Physical Grids

Both the FV dynamical core and the HS physics employ a latitude-longitude grid. FV, however, uses a *staggered* grid on which winds are defined on the boundaries between cells, while in HS all values are defined in the cell center. ESMF provides mechanisms for describing these and other staggerings, as well as regridding software to interpolate values between staggered grids in a physically consistent manner. In Code Example 3, the physical (latitude-longitude) grid for FV is defined; the distributed grid is implied in the decompositions defined in the two dimensions `countsPerDEDecomp1` and `countsPerDEDecomp2`.

Code Example 3: *Creation of an ESMF grid. The global dimensions of the domain, the description of the domain, and interval sizes are specified. The grid is defined as a* latitude-longitude *grid spanning the sphere, which is periodic in longitude. The decomposition is specified by the distributions in both dimensions,* `countsPerDEDecomp1` *and* `countsPerDEDecomp2`. *A grid staggering* `ESMF_GridStagger_D` *is specified for this case where wind speed values are defined on the cell's boundaries, not in its center. An ESMF* grid *is returned.*

```
GRID%GRIDXY = ESMF_GridCreateLogRectUniform(numDims=2, counts = (/grid%im, grid%jm/),       &
  minGlobalCoordPerDim=(/-PI-delta1(1)/2,-PI/2-delta2(1)/2/), deltaPerDim=(/delta1,delta2/),&
  layout=layout, periodic=(/ESMF_TRUE, ESMF_FALSE/), horzGridKind=ESMF_GridKind_LatLon,     &
  horzStagger=ESMF_GridStagger_D, horzCoordSystem=ESMF_CoordSystem_Spherical,               &
  countsPerDEDecomp1=imxy, countsPerDEDecomp2=jmxy, name="FVCORE horizontal grid", rc=status)
```

3.3 Data Decomposition and Redistribution

The FV and HS components distribute their data in fundamentally different manners. HS operations are strictly in the vertical, and therefore it is logical to distribute the data in both latitude and longitude. FV, on the other hand, performs a vertical integration and contains east-west/north-south data dependencies as well. Near the poles the zonal grid intervals become smaller, and the east-west dependencies heavily outweigh the north-south. Thus FV distributes data in latitude and level for most calculations and in latitude and longitude for vertical integration.

While these distributions optimize the parallel performance for the individual components, the data must be redistributed at the component boundaries. As described in Section 2.2, ESMF provides extensive support for regridding data. Code Example 4 illustrates the data redistribution: *routes* (communication patterns) are derived from the XY- and YZ-decomposed fields during the initialization step. These can be applied repeatedly to the fields.

Code Example 4: *Data redistribution between XY and YZ decompositions. ESMF* fields *are created for arrays in each of the two employed decompositions. ESMF* routes `xy_to_yz` *and* `yz_to_xy` *are defined describing the communication pattern between the two fields. These routes can be applied repeatedly when the data needs to be redistributed between the fields.*

```
fieldXY = ESMF_FieldCreate( GRID%GRIDXY, ArrayXY, horzRelloc=ESMF_CELL_CENTER, &
                            haloWidth=hWidth, name="field2", rc=rc )
fieldYZ = ESMF_FieldCreate( GRID%GRIDYZ, ArrayYZ, vertRelloc=ESMF_CELL_CENTER, &
                            haloWidth=hWidth, name="field2", rc=rc )
call ESMF_FieldRedistStore(fieldXY, fieldYZ, delayout, xy_to_yz, rc=rc)
call ESMF_FieldRedistStore(fieldYZ, fieldXY, delayout, yz_to_xy, rc=rc)
  :
call ESMF_FieldRedist(fieldXY, fieldYZ, xy_to_yz, rc=rc)
call ESMF_FieldGetArray(fieldYZ, arrayYZ, rc)
  :
call ESMF_FieldRedist(fieldYZ, fieldXY, yz_to_xy, rc=rc)
call ESMF_FieldGetArray(fieldXY, arrayXY, rc)
```

3.4 FV Subcomponents

The component structure can be recursive, that is, an ESMF component can be constructed from further ESMF components. Such is the case for the finite-volume dynamical core, which consists of a solver for the primitive equations and an algorithm for the advection of tracers (atmospheric constituents, e.g. water vapor or particulates, which are carried by the winds).

The primitive equation (PE) component and the tracer advection (TA) component are connected by couplers which are internal to FV. Just as the main FVHS program creates the FV and HS components and their couplers, the FV `Initialize` method creates the PE and TA components and their couplers.

4 Performance Studies

While FVHS is not a full general circulation model (GCM), its scalability on massively parallel computers gives a good indication of the performance which can be expected of an ESMF-compliant GCM.

Three hour simulations were performed with the FVHS model using a time step of 30 minutes. This is sufficiently long to determine the execution time of one time step in a longer production simulation. In Figure 2 the timings for the FV component are given. Since the Held-Suarez Physics is a much simpler calculation than the dynamics, FV makes up most of the overall computation.

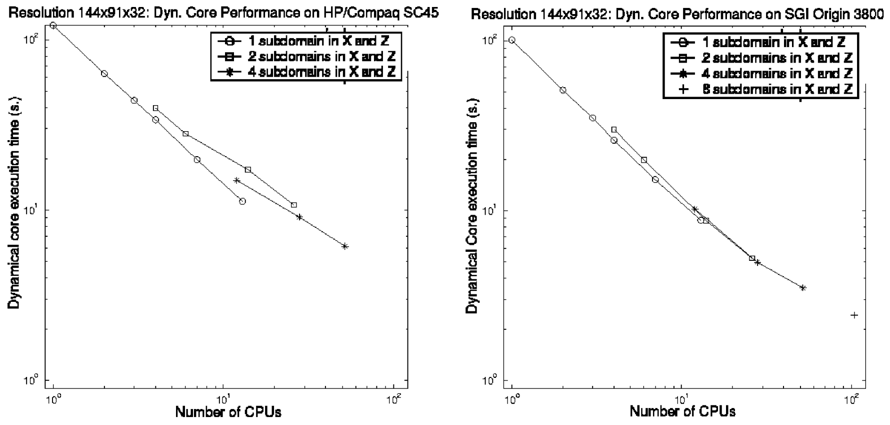


Fig. 2. Even for a relatively low resolution, $2^o \times 2.5^o \times 32$-level problem using 4-byte arithmetic, the ESMF dynamical core component scales adequately on the HP/Compaq 3800 and SGI Origin 3800 at NASA Goddard Space Flight Center. A 3 hr. atmospheric simulation is performed, and the scalability of the FV dynamical core component, which includes communication both in the form of halo exchanges and transposes. A one-dimensional domain decomposition is indicated with an $\bigcirc$. Additional parallelism with two ($\square$) subdomains comes at some additional cost: the redistribution (transpose) must be performed on several arrays for each time step. However, for four ($*$), and eight ($+$) subdomains, there is no additional overhead, and the code scales to roughly 100 CPUs

The results for this "small" problem indicate that the ESMF overhead for each invocation of the `Run` method is scalable. However, they do not make a quantitative statement about the magnitude of the overhead itself. Besides attaining scalability, the goal of the ESMF implementation is naturally to retain the performance of the underlying application.

In order to determine the introduced overhead, timings of the FV dynamical core [11] in the existing Community Atmosphere Model (CAM) were made. This is unfortunately not a precise comparison, since (1) the CAM grid has 26 vertical levels instead of 32 for FVHS, and (2) calculations are performed with 8-byte arithmetic in the former instead of the 4-byte arithmetic used in the latter. These differences are imposed by the data sets which are used for the respective CAM and FVHS runs. Experience shows, however, that the performance gains by the fewer level calculations is roughly compensated by the expense of the double precision calculation. The results are illustrated in Figure 3 in fact imply that the ESMF overhead is well below the 10% allowed by the ESMF requirements document [1].

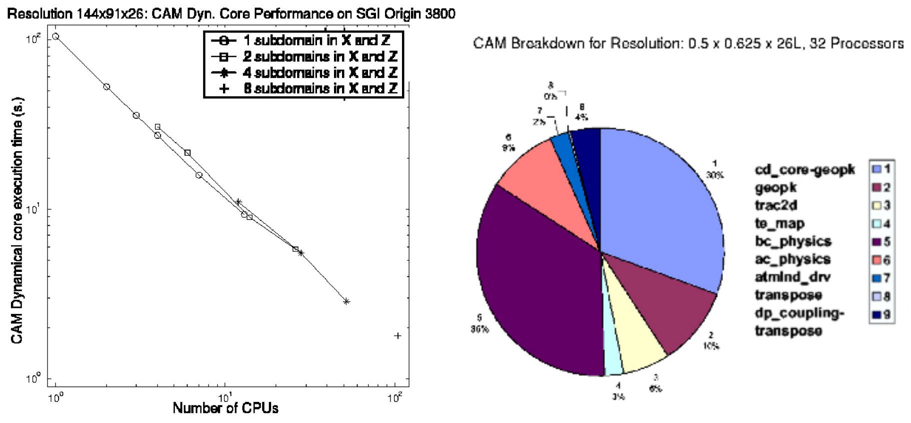


Fig. 3. The graph on the left side illustrates the performance of the dynamical core (not ESMF-compliant) at $2^o \times 2.5^o \times 26$ levels with 8-byte arithmetic in the Community Atmosphere Model on the SGI Origin 3800. The meaning of the symbols is identical to the corresponding SGI graph in Figure 2. The pie chart at right indicates the breakdown of times for the components of the full CAM, when run at $0.5^o \times 0.625^o \times 26$ level resolution. The dynamical core comprises slices 1, 2, 3 and 8, and accounts for nearly half of the overall computation

5 Summary of Experiences

The components mentioned in Section 3 predate ESMF, thus the codes had to be rewritten for ESMF compliance. The fundamental design of the original software lent itself well to the division into `Initialize`, `Run` and `Finalize` methods, thus the code restructuring was kept to a minimum.

In spite of this, there were impediments to a quick implementation. First, the import and export states of the FV dynamical core were not entirely appropriate for the Held-Suarez component. Normally the mapping to the required FV import state and from the HS export state would have been programmed in the couplers, but in this case it was decided to modify the import and export states themselves, bringing them in line with other dynamical cores. We suspect it is normal that existing software components will sometimes require revisions in the import and export states to make them general enough for use in other ESMF applications. The ensuing development overhead unfortunately cannot be alleviated by a software framework.

Secondly, some planned ESMF functionality needed in FVHS had not been implemented when FVHS was first integrated. Thus we wrote our own temporary implementations of the regrid interpolations between staggered and unstaggered grids, I/O operations, and other utilities. These deficiencies have since been corrected by the ESMF core team. On the other hand, the framework-specific tasks, such as the integration of components through couplers, and creation of physical and decomposition grids, were straightforward and took little time.

The performance of the FV component proved to be comparable to that of the original dynamical core, indicating that the overhead an ESMF-compliant application was in this case not significant.

Acknowledgments

The NASA ESTO program supports the development of ESMF through contracts under NASA Cooperative Agreement Notice CAN-00-OES-01.

References

1. V. Balaji, T. Bettge, B. Boville, T. Craig, C. Cruz, A. da Silva, C. DeLuca, B. Eaton, R. Hallberg, C. Hill, M. Iredell, R. Jacob, P. Jones, B. Kauffman, J. Larson, J. Michalakes, D. Neckels, J. Rosinski, S. Smithline, M. Suarez, J. Wolfe, W. Yang, M. Young, L. Zaslavsky. Earth System Modeling Framework; ESMF Requirements. `http://www.esmf.ucar.edu/esmf_docs/ESMF_last_reqdoc/`.
2. V. Balaji, B. Boville, N. Collins, T. Craig, C. Cruz, A. da Silva, C. DeLuca, R. Hallberg, C. Hill, M. Iredell, R. Jacob, P. Jones, B. Kauffman, E. Kluzek, J. Larson, J. Michalakes, D. Neckels, W. Sawyer, E. Schwab, S. Smithline, M. Suarez, B. Womack, W. Yang, M. Young, L. Zaslavsky. Earth System Modeling Framework; DRAFT ESMF Architecture. `http://www.esmf.ucar.edu/esmf_docs/ESMF_archdoc/`.
3. M. B. Blackmon, B. Boville, F. Bryan, R. Dickinson, P. Gent, J. Kiehl, R. Moritz, D. Randall, J. Shukla, S. Solomon, G. Bonan, S. Doney, I. Fung, J. Hack, E. Hunke, and J. Hurrell. The Community Climate System Model. *BAMS*, 82(11):2357–2376, 2001.
4. Geophysical Fluid Dynamics Laboratory. The Flexible Modeling System. `http://www.gfdl.noaa.gov/fms`.
5. I. Held and M. Suarez. A Proposal for the Intercomparison of the Dynamical Cores of Atmospheric General Circulation Models. *BAMS*, 75(10):1825–1830, 1994.
6. C. Hill, A. Adcroft, D. Jamous, and J. Marshall. A strategy for tera-scale climate modeling. In *Proc. 9th ECMWF Workshop on the Use of HPC in Meteorology*. World Scientific Press, 2001.
7. C. Hill, C. DeLuca, Balaji, M. Suarez, and A. da Silva. The Architecture of the Earth System Modeling Framework. *Computing in Science & Engineering*, 6(1):18–28, 2004.
8. J. Larson, R. Jacob, I. Foster, and J. Guo. The Model Coupling Toolkit. In *Proc. Int'l Conf. Computer Science*, pages 185–194. Springer-Verlag, 2001. Lecture Notes in Computer Science 2073.

9. S.-J. Lin and R. B. Rood. A 'vertically Lagrangian' Finite-Volume Dynamical Core for Global Models. *Monthly Weather Review*, 2004. Submitted for publication.
10. J. Michalakes, S. Chen, J. Dudhia, L. Hart, J. Klemp, J. Middlecoff, and W. Skamarock. Development of a next generation regional weather research and forecast model. In *Proc. 9th ECMWF Workshop on the Use of HPC in Meteorology*. World Scientific Press, 2001.
11. W. Sawyer and A. Mirin. A scalable implementation of a finite-volume dynamical core in the Community Atmosphere Model. In *Proc. Parallel and Distributed Computing and Systems*, ACTA Press, 2004. Accepted for publication.
12. W. Sawyer and P. Messmer. Parallel Grid Manipulations for General Circulation Models. In *Parallel Processing and Applied Mathematics : 4th International Conference*, pages 564–571. Springer-Verlag, 2002. Lecture Notes in Computer Science 2328.
13. D. Schaffer and M. Suarez. Design and Performance Analysis of a Massively Parallel Atmospheric General Circulation Model. *J. Sci. Programming*, 8:49–57, 2000.

Parallel Discrete Event Simulations of Grid-Based Models: Asynchronous Electromagnetic Hybrid Code[*]

Homa Karimabadi[1], Jonathan Driscoll[1], Jagrut Dave[1,2], Yuri Omelchenko[1], Kalyan Perumalla[2], Richard Fujimoto[2], and Nick Omidi[1]

[1] SciberNet, Inc., Solana Beach, CA, 92075, USA
{homak,driscoll,yurio,omidi}@scibernet.com
[2] Georgia Institute of Technology, Atlanta, GA, 30332, USA
{jagrut,kalyan,fujimoto}@cc.gatech.edu

Abstract. The traditional technique to simulate physical systems modeled by partial differential equations is by means of a time-stepped methodology where the state of the system is updated at regular discrete time intervals. This method has inherent inefficiencies. Recently, we proposed [1] a new asynchronous formulation based on a discrete-event-driven (as opposed to time-driven) approach, where the state of the simulation is updated on a "need-to-be-done-only" basis. Using a serial electrostatic implementation, we obtained more than two orders of magnitude speedup compared with traditional techniques. Here we examine issues related to the parallel extension of this technique and discuss several different parallel strategies. In particular, we present in some detail a newly developed discrete-event based parallel electromagnetic hybrid code and its performance using conservative synchronization on a cluster computer. These initial performance results are encouraging in that they demonstrate very good parallel speedup for large-scale simulation computations containing tens of thousands of cells, though overheads for inter-processor communication remain a challenge for smaller computations.

1 Introduction

Computer simulations of many important complex physical systems have reached a barrier as existing techniques are ill-equipped to deal with the multi-physics, multi-scale nature of such systems. An example is the solar wind interaction with the Earth's magnetosphere. This interaction leads to a highly inhomogeneous system consisting of discontinuities and boundaries and involves coupled processes operating over spatial and temporal scales spanning several orders of magnitude. Inclusion of such disparate scales is beyond the scope of existing codes [2].

We have taken a new approach [1] to the simulation of such complex systems. The conventional time-stepped grid-based Particle-In-Cell (PIC) models provide the sequential execution of synchronous (time-driven) field and particle updates. In a synchronous simulation, the distributed field cells and particles undergo simultaneous state transitions

[*] Research was supported by NSF ITR grant 0325046 at SciberNet Inc. and 0326431 at Georgia Institute of Technology. Some of the computations were performed at the San Diego Supercomputing Center.

at regular discrete time intervals. In contrast, we propose a new, asynchronous type of PIC simulation based on a discrete-event-driven (as opposed to time-driven) approach, where particle and field time updates are carried out in space on a “need-to-be-done-only” basis. In these simulations, particle and field information “events” are queued and continuously executed in time. The technique has some similarity to Cellular Automata (CA) in that complex behaviors result from interaction of adjacent cells [3]. However, unlike CA, the interactions between cells are governed by a full set of partial differential equations rather than the simple rules as are typically used in CA. The power of this technique is in its asynchronous nature as well as elimination of unnecessary computations in regions where there is no significant change in time. This is in contrast to CA, which are largely based on synchronous execution (e.g. [4]); to date, asynchronous parallel discrete event simulation of CA have only been applied to relatively simple phenomena such as Ising spin [5]).

Using a serial electrostatic model, we have shown [1] that the discrete event technique can lead to more than two orders of magnitude speedup compared to conventional techniques. In the following, we discuss issues associated with the extension of this technique to parallel architectures. We then demonstrate, through a newly developed parallel hybrid code, that parallel processing can provide an additional order of magnitude improvement in performance.

2 Parallel Computation Issues

Discrete Event Simulation (DES) offers substantial benefits compared to conventional explicit time-driven simulation by reducing the amount of computation that must be performed. However, by itself, DES is not sufficient to achieve the desired performance and scalability. Parallel DES (PDES) can help address this issue. However, the irregular nature of PDES computations leads to difficulties. Synchronization overhead, the number of concurrent computations, load distribution and event processing rate impact PDES performance significantly [6].

As in conventional (time-driven) simulations, the parallelization of asynchronous (event-driven) continuous PIC models is realized by decomposing the global computation domain into subdomains. In each subdomain, individual cells and particles are aggregated into containers that may be mapped to different processors. The parallel execution of time-driven simulations is commonly achieved by copying field information from the inner lattice cells to ghost cells of neighboring subdomains and exchanging out-of-bounds particles between the processors at the end of each update cycle. By contrast, in parallel asynchronous PIC simulations both particle and field events are not synchronized by the global clock (i.e. they do not take place at the same time intervals throughout the simulation domain), but occur at arbitrary time intervals, introducing synchronization problems. Unless precautions are taken, a process may receive an event message from a neighbor with a simulation time stamp that is in its past.

In the following, we assume that the parallel simulation is composed of a collection of Simulation Processes (SPs) that communicate by exchanging time stamped event messages. Broadly, synchronization approaches may be classified as *conservative* or *optimistic*. Conservative synchronization ensures that each simulation process never receives an event in its past [8, 9]. Runtime performance is critically dependent on *apriori* determination of an application property called *lookahead (a time interval)*, which is roughly dependent on the degree to which the computation can predict future interactions with other processes without global information. On the other hand, the optimistic approach allows a process to receive a message in its past, but uses a rollback mechanism to recover [7]. Further discussion can be found in [10,11,12].

Another important issue concerns load balancing. As with any parallel or distributed application, the computation must be evenly balanced across processors and interprocessor communication should be minimized to achieve the best performance. Often these are conflicting goals. This is particularly challenging in PDES because of its irregular, unpredictable nature. Load balancing can greatly affect the efficiency of synchronization mechanisms (e.g. poor load distribution can lead to excessive rollbacks in optimistic systems). Automated schemes that balance workload at runtime using process migration present new challenges in this area [13,15].

Finally, it is desirable to decouple the parallel simulation engine that handles synchronization and communication from the application/models. This reduces the burden of the application developer, by not requiring an understanding of underlying PDES synchronization mechanisms. We have used an extensible simulation engine that provides multiple synchronization and event delivery mechanisms through a single interface, named μsik [16].

3 DES Model

We have developed a general architecture for parallel discrete event modeling of grid-based models. Details will be presented elsewhere. Here we present a simplified version of our technique that illustrates the salient features of our model without getting bogged down in all the details. In the following, we show results from a 1D parallel hybrid code (*light* version) that we have developed and tested using μsik as the simulation engine. This code is used to highlight unique parallel issues that are encountered in the DES modeling of plasmas. The light version does not strictly conserve flux. However, the lack of strict local flux conservation does not change the result significantly in the problem of interest here.

3.1 Hybrid Algorithm

Electromagnetic hybrid algorithms with fluid electrons and kinetic ions are ideally suited for physical phenomena that occur on ion time and spatial scales. Maxwell's equations are solved by neglecting the displacement current in Ampere's law (Darwin approximation), and by explicitly assuming charge neutrality. There are several variations of electromagnetic hybrid algorithms with fluid electrons and kinetic ions [18]. Here we use the one-dimensional resistive formulation [19] which casts field equations in terms of vector potential. The model problem uses the piston method where incoming plasma moving with flow speed larger than its thermal speed is reflected off the piston located on the rightmost boundary. This leads to the generation of a shockwave that propagates to the left. In this example, we use a flow speed large enough to form a fast magnetosonic shock. In all the runs shown here, the plasma is injected with a velocity of 1.0 (normalized to upstream Alfven speed), the background magnetic field is tilted at an angle of 30^o, and the ion and electron betas are set to 0.1.

The simulation domain is divided into cells [1], and the ions are uniformly loaded into each cell. We conducted experiments ranging from 4,096 to 65,536 cells, and initialized each simulation to have 100 ions per cell. Each cell is modeled as an SP in μsik and the state of each SP includes the cell's field variables. The main tasks in the simulation are to a) initialize fields, b) initialize particles, c) calculate the exit time of each particle, d) sort IonQ (see below), e) push particle, f) update fields, g) recalculate exit time, and h) reschedule. This is accomplished through a combination of priority queues and three main classes of events. The ions are stored in either one of two priority queues as illustrated in Fig. 1. Ions are initialized within cells in an IonQ. As ions move out of the left most cell, new ions are injected into that cell in order to keep the flux of incoming ions fixed at the left boundary. MoveTime is the time at which an ion is to be moved next. The placement and removal of ions in IonQ and PendQ is controlled by comparing their MoveTimes to the current time and lookahead. Ions with MoveTimes more than current time + 2*lookahead have not yet been scheduled and are kept in the IonQ. A wakeup occurs when the fields in a given cell change by more than a certain threshold and MoveTimes of particles in the cell need to be updated. On a wakeup, only the ions in this queue recalculate their MoveTimes. Because ions in the IonQ have not yet been scheduled, a wakeup requires no event retractions. If an ion's MoveTime becomes less than current time + 2*lookahead in the future, the ion is scheduled to move, and is removed from the IonQ and placed in the PendQ. Thus, the front of the IonQ is at least one lookahead period ahead of the current time. This guarantees that each ion move will be scheduled at least one lookahead period in advance. The PendQ is used to keep track of ions that have already been scheduled to exit, but have not yet left the cell. These particles have MoveTimes that are less than the current time. Ions in the PendQ with MoveTimes earlier than the current time have already left the cell and must be removed before cell values such as density and temperature are calculated.

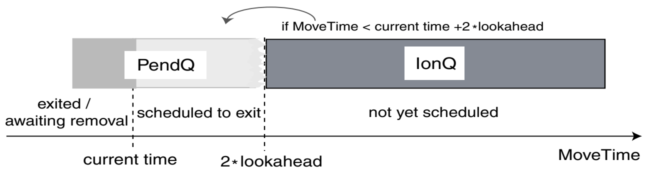


Fig. 1. At any moment, the ions in a cell are stored in one of two queues, the IonQ and the PendQ. Both are priority queues, sorted so that the ion with the earliest exit time is at the top

Events can happen at any simulation time and are managed separately by individual cells of the simulation. The flow of the program including functions of events and their interaction with μsik is illustrated in Fig. 2. In this simulation, each cell handles three different types of events.

<u>SendIon Event</u>: This event is first run on each cell when the simulation is initialized,

and is responsible for sending ions from one cell to the next. This is accomplished by scheduling the complementary "AddIon" events for neighboring cells. The SendIon event schedules an AddIon event corresponding to every Ion which exits within two lookahead periods, and always schedules at least one SendIon event. In addition, the SendIon Event checks to see if the fields have changed by some tolerance, waking up particles in that cell if necessary. SendIon events occur frequently and as a whole are computationally significant.

AddIon Event: This event is used to add a single ion to a cell. The ion's new exit time is calculated, and then it is added to the IonQ. The fields in the cell are then updated and Notify Events are scheduled for the left and right neighbor cells to inform them of the field change. The AddIon Event causes state changes and occurs sporadically in large batches.

NotifyEvent: This event updates the vector potential and temperature for the two neighbors.

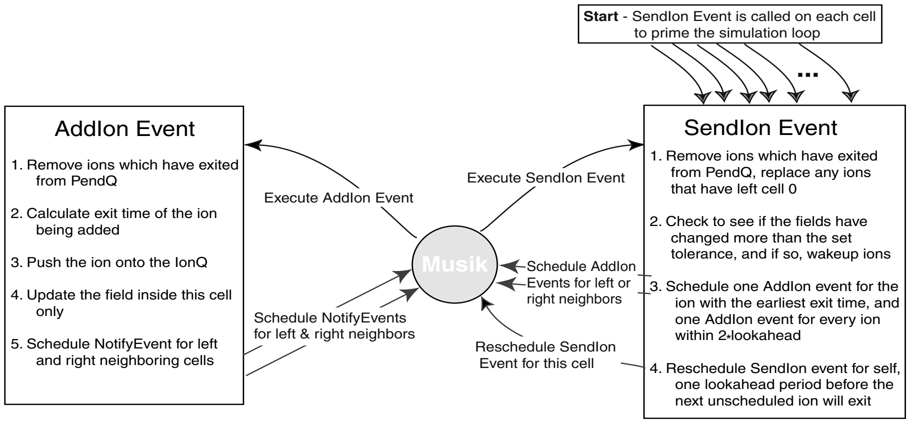
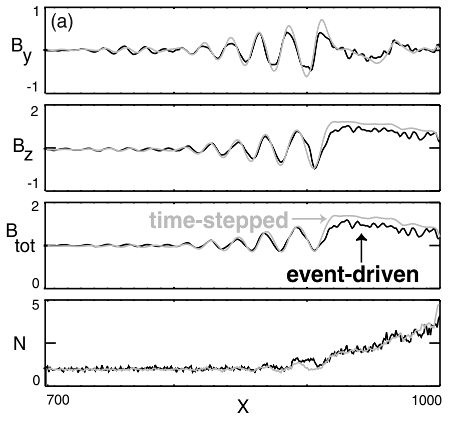

Start - SendIon Event is called on each cell to prime the simulation loop
...
AddIon Event
1. Remove ions which have exited from PendQ
2. Calculate exit time of the ion being added
3. Push the ion onto the IonQ
4. Update the field inside this cell only
5. Schedule NotifyEvent for left and right neighboring cells
Execute AddIon Event
Execute SendIon Event
Musik
Schedule AddIon Events for left or right neighbors
Schedule NotifyEvents for left & right neighbors
Reschedule SendIon Event for this cell
SendIon Event
1. Remove ions which have exited from PendQ, replace any ions that have left cell 0
2. Check to see if the fields have changed more than the set tolerance, and if so, wakeup ions
3. Schedule one AddIon event for the ion with the earliest exit time, and one AddIon event for every ion within 2*lookahead
4. Reschedule SendIon event for self, one lookahead period before the next unscheduled ion will exit

Fig. 2. Flow diagram of the parallel hybrid code

Exit Time. We take the electric and magnetic fields to be constant within a cell, with arbitrary orientation and magnitude. In this case, a charged particle will have an equation of motion that can be calculated analytically and has the general form $\mathbf{R}(t) = \mathbf{A}t^2 + \mathbf{B}t + \mathbf{r}_c \sin(\omega_c t + \phi) + \mathbf{C}$, where $R(t)$is the position of the particle. Newton's method is used to solve for the exact exit time.

Lookahead. If the typical velocity of a particle is v, and a typical cell width is x, then the time it takes for a particle to cross a cell is x/v. Lookahead must be a factor smaller than this time so that a particle covers a small fraction of the cell width in one lookahead period. On the other hand, if the lookahead is too small, the parallel performance will be poor. This happens when there are few event computations during a lookahead period. Synchronization overhead becomes larger than the computational load. We use the time it takes for the first particle to exit a cell to set the lookahead.



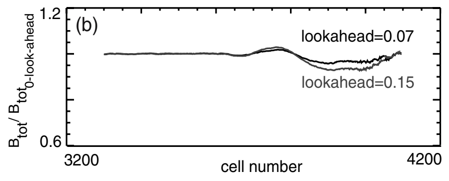


Fig. 3. (a) Comparison of time-stepped and event-driven simulations of a fast magnetosonic shock. (b) The ratio of the total magnetic field from lookahead runs of 0.07 and 0.15 relative to the field from zero lookahead run

4 Results

Figure 3(a) compares results of traditional time-stepped hybrid simulation and our event-stepped simulation for a single processor. We have plotted the y and z (transverse) components of the magnetic field, the total magnetic field, and the plasma density versus x, after the shock wave has separated from the piston on the right hand side. The match between the two simulations is remarkable as DES captures the (i) correct shock wave speed, and (ii) details of the wavetrain associated with the shock wave. This match is impressive considering the fact that the differences seen in Fig. 3(a) are within statistical fluctuations associated with changes in the noise level in hybrid codes.

4.1 Effect of Lookahead

Next, we consider the effects of changing the lookahead on both the accuracy of the results as well as the execution time. The hardware for the runs shown here was a high-performance cluster at the Georgia Tech High Performance Computing laboratory. The cluster has 8 nodes, each with 4 2.8 GHz Xeon processors and 4 GB of RAM. The simulation uses 4 μsik Federates (one per processor), each with 2 Regions and 512 cells per Region. A Region is a grouping of cells for efficient load-distribution, described in Sect. 4.3.

Figure 3(b) shows variations in the spatial profile of B_{tot}, with lookahead. The zero lookahead run yields the most accurate result and is treated as a baseline. In the hybrid algorithm, the maximum lookahead must be less than the exit time of the earliest scheduled particle, which is approximately 0.15 for our choice of parameters. Deviations of the profile from the baseline are less than 10%, even when the maximum lookahead value is used.

Figure 4(a) shows the speedup in execution time relative to the zero lookahead run. The important point from this figure is that even small departures from zero lookahead lead to substantial improvements in speed. In fact, the most dramatic speedup (a factor of 3) is achieved when lookahead is changed from 0 to 0.005. Further changes in lookahead do improve performance, but at a much slower rate. For example, increasing the lookahead by an order of magnitude from 0.005 to 0.05 leads to only an additional 15% speedup.

4.2 Scaling with the Number of Processors

The 1D modeling of the shock problem in Fig. 3(a) can be easily performed with a serial version of our code. However, our ultimate goal is to develop a 3D version of the code. As a simple means to evaluating the parallel execution in a 3D model, we have considered cell numbers as large as 65,536. This is sufficient to identify the key issues of parallel execution. Figure 4(b) shows the speedup as a function of the number of processors up to 128. The speedup is measured with respect to a sequential run. These runs were made on a 17 node cluster, with each node having 8 550 MHz CPUs and 4 GB of RAM. The simulation domain consisted of 8,192 cells in one case and 65,536 cells in the other.

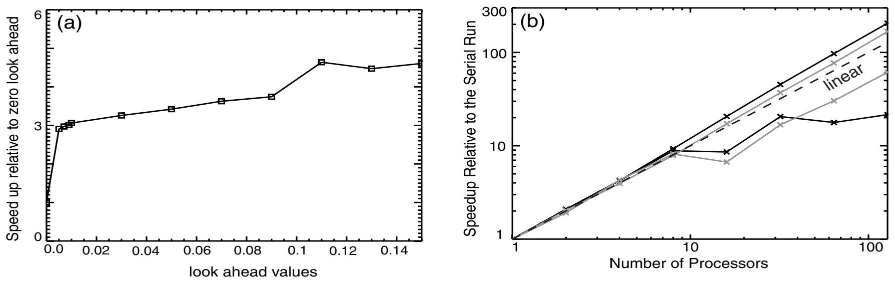


Fig. 4. (a) Speedup with increased lookahead. (b) Scaling with the number of processors. The dashed line is a linear scaling curve. Speedup for 8,192 cells is in black and 65,536 cells is in gray. For each domain size, there are two curves - speedup considering the overall execution time and speedup without considering communication time. The scaling is superlinear if communication time is removed

As is evident from Fig. 4(b), the parallel speedup is good till eight processors, but declines as more processors are added. This is due to the architecture of the cluster, which uses a collection of 8 processor computers communicating through TCP/IP. With up to 8 processors, the entire simulation runs through shared memory, and the communication overheads are low. However, with more than 8 processors, the overheads associated with TCP/IP begin to offset the speed gained by using more processors. This reduces the slope of the curve. For 8,192 cells, the speedup does not increase significantly after 32 processors. This is because of increased synchronization costs, which negate the gains from parallel processing. Processors do not have a sufficient computational load between global synchronizations and spend a greater fraction of time waiting on other processors. For 65,536 cells, there is enough computation between global synchronizations to obtain good speedup up to 128 processors.

Since the overheads associated with inter-processor communication become relatively smaller as the simulation size increases, we do not anticipate this effect being as pronounced with larger, 3D simulations. In 3D simulations, each processor would have several orders of magnitude more cells, making the relative overheads associated with TCP/IP much less. To test the idea that poor scaling is the result of the communication overheads, we have also plotted speedup with the communication time subtracted out in Fig. 4(b). The speedup in this case is better than expected, and is in fact super-linear. In other words, doubling the number of processors more than doubles the execution speed. This is the result of a peculiar feature of DES. Execution time does not necessarily scale linearly with the simulation size, even on one processor. This is in part due to the fact that as the event queue becomes larger (contains more pending events), the time associated with scheduling and retrieving each event increases. So, as the simulation gets distributed over more and more processors, each processor is effectively dealing with a smaller piece of the simulation, making the scaling non-linear. Memory performance (specifically, cache performance) can also lead to super-linear speedup. By keeping the same size problem but distributing it over more processors, the memory footprint in each processor shrinks. The total amount of cache memory increases in proportion with the number of processors used - with enough processors, one can, for example, fit the entire computation into the processors' caches. An extreme case of this is when the problem is so large that it does not fit into the memory of a single machine, causing excessive paging. Although both effects could be causing the super-linear scaling seen in Fig. 4(b), the data structure performance appears to be dominant. In a no-load test of μsik, we changed the number of cells from ten to a million. The time to process a single event increased from 6.50 to 22.15 microseconds, indicating a scaling behavior of NlogN.

Figure 5(a) shows the percentage of time spent in communication and blocking in each case. There is a significant increase in the fraction of time spent in communication and blocking for more than 8 processors. For 65,536 cells, the percentage settles to around 60% for higher number of processors. However, for 8,192 cells, the percentage keeps on increasing until 90% for 128 processors.

4.3 Load Balancing

Figure 5(b) shows the variation in execution time as a function of the number of Regions per processor, as distributed by the Region Deal algorithm. In this scheme, the simulation is broken into small Regions which are then "dealt" out much like a card game among processors. These simulation runs were performed on the first cluster mentioned earlier.

The curves show a different trend for higher number of processors (4,8,16) than for 2 processors. For higher number of processors, the variation in execution time because of the Region Deal load balancing scheme is less pronounced. The best execution times are close to the execution time for 1 Region per processor, with variations of less than 1,000 seconds. Also, increasing the number of Regions per processor increases the execution time in most cases. This is because of the increased synchronization overhead that negates the benefits of the load distribution scheme. For 2 processors, having more

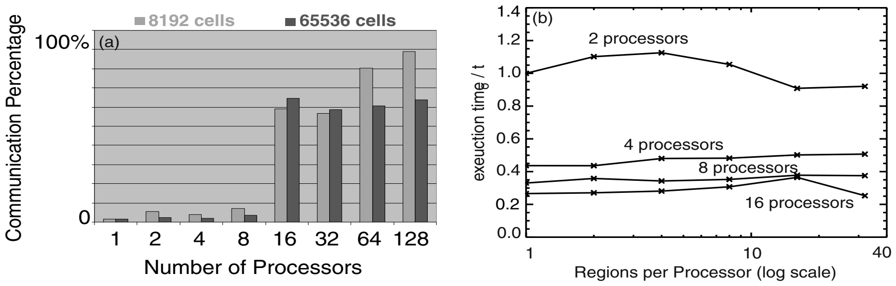


Fig. 5. (a) Percentage of time spent in communication. (b) Performance of load balancing algorithm

Regions per processor leads to better load distribution and hence reduced execution time. The execution time settles around 14,000 seconds for 16 Regions or more. In this case too, contiguous cells are assigned to different processors and incur greater synchronization overhead for higher number of Regions per processor.

References

1. Karimabadi, H, Driscoll, J, Omelchenko, Y.A. and N. Omidi, *A New Asynchronous Methodology for Modeling of Physical Systems: Breaking the Curse of Courant Condition, J. Computational Physics,* (2005), in press.
2. Karimabadi, H. and N. Omidi. *Latest Advances in Hybrid Codes and their Application to Global Magnetospheric Simulations*. in *GEM, http://www-ssc.igpp.ucla.edu/gem/tutorial/index.html)* (2002).
3. Ilachinski, A., Cellular Automata, A Discrete Universe, World Scientific, 2002.
4. Smith, L., R. Beckman, et al. (1995). TRANSIMS: Transportation Analysis and Simulation System. Proceedings of the Fifth National Conference on Transportation Planning Methods. Seattle, Washington, Transportation Research Board.
5. Lubachevsky, B. D. (1989). Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks. Communications of the ACM **32**(1): 111-123.
6. Fujimoto, R.M., *Parallel and Distributed Simulation Systems*. (2000): Wiley Interscience.
7. Jefferson, D., *Virtual Time,* ACM Transactions on Programming Languages and Systems, (1985), 7(3):pp. 404-425.
8. Chandy, K. and J. Misra (1979). Distributed Simulation: A case study in design and verification of distributed programs. IEEE Transactions on Software Engineering.
9. Chandy, K. and J. Misra (1981). Asynchronous distributed simulation via a sequence of parallel computations. Communications of the ACM. **24.**
10. Fujimoto, R. M. (1999), Exploiting Temporal Uncertainty in Parallel and Distributed Simulations, Proceedings of the 13^{th} Workshop on Parallel and Distributed Simulation: 46-53.
11. Rao, D. M., N. V. Thondugulam, et al. (1998). Unsynchronized Parallel Discrete Event Simulation. Proceedings of the Winter Simulation Conference**:** 1563-1570.
12. Rajaei, H., R. Ayani, et al. (1993). The Local Time Warp Approach to Parallel Simulation. Proceedings of the 7th Workshop on Parallel and Distributed Simulation**:** 119-126.
13. Boukerche, A., and S. K. Das, Dynamic Load Balancing Strategies for Conservative Parallel Simulations, Workshop on Parallel and Distributed Simulation, 1997.

14. Carothers, C. D., and R. M. Fujimoto, “Efficient Execution of Time Warp Programs on Heterogeneous, NOW Platforms,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 11, No. 3, pp. 299-317, March 2000.
15. Gan,B. P., *et al.*, Load balancing for conservative simulation on shared memory multiprocessor systems, Workshop on Parallel and Distributed Simulation, 2000
16. Perumalla, K.S., *μsik – A Micro-Kernel for Parallel and Distributed Simulation Systems*, to appear in the Workshop on Principles of Advanced and Distributed Simulation, May, 2005.
17. Bagrodia, R., R. Meyer, et al. (1998). Parsec: A Parallel Simulation Environment for Complex Systems. IEEE Computer **31**(10): 77-85.
18. Karimabadi, H., D. Krauss-Varban, J. Huba, and H. X. Vu, On magnetic reconnection regimes and associated three-dimensional asymmetries: Hybrid, Hall-less hybrid, and Hall-MHD simulations, J. Geophys. Res., Vol. 109, A09205,(2004).
19. Winske, D. and N. Omidi, *Hybrid codes: Methods and Applications*, in *Computer Space Plasma Physics: Simulation Techniques and Software*, H. Matsumoto and Y. Omura, Editors. (1993), Terra Scientific Publishing Company. p. 103-160.

Electromagnetic Simulations of Dusty Plasmas

Peter Messmer

Tech-X Corporation
5621 Arapahoe Avenue, Suite A, Boulder, CO 80303, USA
messmer@txcorp.com

Abstract. Dusty plasmas are ionized gases containing small particles of solid matter, occurring e.g. in space, fusion devices or plasma processing discharges. While understanding the behavior of these plasmas is important for a variety of applications, an analytic treatment is difficult and numerical methods are required. Ideally, the dusty plasma would be treated kinetically, modeling each particle individually. However, this is in general far too expensive and approximations have to be used, like representing the electrons as fluids or treating Maxwell's equations in the electrostatic limit. Here we report on fully kinetic electromagnetic simulations of dusty plasmas, using the parallel kinetic plasma simulation code VORPAL. A brief review of the particle-in-cell (PIC) algorithm, including the model for modeling dust grains, and first results of single dust grain simulations are presented.

1 Introduction

Dust - plasma interactions play an important role in space plasma systems. Comets, rings in planetary magnetospheres, exposed dusty surfaces, or the zodiacal dust cloud are all examples where dusty plasma effects shape the size and spatial distribution of small dust particles. Simultaneously, dust is often responsible for the composition, density and temperature of its plasma environment. The dynamics of charged dust particles can be surprisingly complex, fundamentally different from the well understood limits of gravity dominated motion (vanishing charge-to-mass ratio) or the adiabatic motion of electrons and ions.

Dust particles in plasmas are unusual charge carriers. They are many orders of magnitude heavier than any other plasma particles and they can have orders of magnitude larger (negative or positive) time-dependent charges. Dusty plasma systems are characterized by new spatial and temporal scales (grain radius, average distance between grains, dust gyro-radius and frequency, etc.), in addition to the customary scales (Debye length, plasma frequency, etc.). Dust particles can communicate non-electromagnetic effects (gravity, drag, radiation pressure) to the plasma, representing new free energy sources. Their presence can influence the collective plasma behavior, altering the wave modes and triggering new instabilities.

2 PIC Algorithm

An efficient way to model kinetic plasmas self-consistently is the well-established particle-in-cell (PIC) algorithm [1,2], which follows the trajectories of a representative number of particles, so called macro-particles, in the electromagnetic field. In the following, we briefly review this algorithm:

The relativistic motion of charged particles in an electromagnetic field is governed by the Newton-Lorentz law

$$\frac{d\gamma v}{dt} = \frac{q}{m}\left(E + \frac{v}{c} \times B\right) \qquad \frac{dx}{dt} = v \tag{2.1}$$

where x, v and q/m are position, velocity and charge to mass ratio of each particle, $\gamma = (1 - v^2/c^2)^{-1/2}$ and c is the speed of light. The evolution of the electric field E and the magnetic inductivity B is determined by Maxwell's equations,

$$\nabla \cdot E = \frac{\rho}{\epsilon_0} \tag{2.2}$$

$$\frac{\partial E}{\partial t} = c^2 \nabla \times B - \frac{j}{\epsilon_0} \tag{2.3}$$

$$\nabla \cdot B = 0 \tag{2.4}$$

$$\frac{\partial B}{\partial t} = -c\nabla \times E \tag{2.5}$$

where j denotes the current density and ϵ_0 is the permittivity of free space.

To simulate a plasma, the equation of motion for the particles and Maxwell's equations have to be solved concurrently. In the PIC algorithm, the field quantities E and B are therefore discretized in space, allowing to use finite differences for the curl operators in Eqs. (2.3) and (2.5). The particles, on the other hand, are located anywhere in the computational domain.

Starting from an initial distribution of particles and fields, the system is evolved in time. Each time-step consists of two alternating phases: In the *particle push* phase, the new particle positions and velocities are determined according to Eq. (2.1). In the following *field solve* phase, the fields are updated using the new particle positions.

In order to accelerate the particles in the particle push, the field quantities have to be determined at the particle location, using e.g. linear interpolation. The actual time integration is then done by a leap-frog scheme.

To update the fields, a variety of algorithms and approximations can be used. Very common for dusty plasma simulations is the electrostatic approximation, which neglects the time dependency of B. The electric field is then determined via Poisson's equation, Eq. (2.2), using the charge density derived from the particle positions.

For systems with fast particles, or to model the interaction with electromagnetic waves, the time dependency of B cannot be neglected. The full set of Maxwell's equations has then to be solved. The fields are coupled to the particles through the current density term in Eq. (2.3) and the charge density term in Eq. (2.2). Different algorithms are known to determine the current density from the particle motion. One example are charge-conserving current deposition algorithms [3], which determine the current flux through a cell surface from the charge carried across this surface due to the particle motion in one time step, have the nice property that they satisfy the continuity equation,

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot j. \tag{2.6}$$

The advantage of this algorithm is that Eq. (2.2) remains satisfied if it was satisfied initially, as can be seen by taking the divergence of Eq. (2.3) and applying the continuity equation,

$$\frac{\partial}{\partial t} \nabla E = -\nabla \frac{j}{\epsilon_0} = \frac{\partial \rho}{\partial t}. \tag{2.7}$$

Therefore, if an electromagnetic PIC code with charge-conserving current deposition is started with a self-consistent solution of E and B, the evolution of the fields is entirely determined by Eq. (2.3) and (2.5). The particle charge is only needed to determine the current density, and no explicit solution of Poisson's equation is required .

The field quantities are placed at different positions in space: The E field components are defined on the cell edges, whereas the B field components are known on the cell surfaces. The components are staggered in a so-called Yee grid [4]. In this setup, e.g. the B_z, E_x and E_y components reside on the same z-plane. This allows to compute the z-component of $\nabla \times E$ at the location of B_z by means of finite-differences. Analogously, all other curl operators can be computed. Providing the current flux at locations of the E field grid, the B and E fields can be updated by leap-frog-scheme.

All finite difference operators are time and space centered to guarantee second order accuracy and the whole scheme requires only local information, making it very suitable for parallel implementations. A drawback is, however, that the timestep Δt and the grid cell spacing, Δx, Δy, Δz, are coupled via a CFL condition

$$c\Delta t \leq \left(\Delta x^{-2} + \Delta y^{-2} + \Delta z^{-2}\right)^{-1/2} \tag{2.8}$$

which states that the propagation of light waves has to be resolved. This can be very restrictive, especially if the main focus of the simulation is on the particle behavior and the particle velocity is much smaller than the speed of light.

2.1 Parallel PIC Algorithm

The PIC algorithm per-se is inherently parallel, as the same procedures (move particles under the influence of Newton-Lorentz force, deposit the particle charge or currents to the field) have to be performed for a large number of particles.

In the electrostatic case, parallelization is complicated due to the solution of Poisson's equation, which requires global information. Highly optimized iterative methods can be used to solve this problem efficiently. The parallelization of the electromagnetic problem turns out to be simpler than the electrostatic case. As only local information is required to update the fields and the particles, parallelization via domain decomposition can lead to good speedup.

2.2 Parallel Plasma Simulation Code VORPAL

VORPAL [5] is a plasma simulation framework under development at the University of Colorado (CU) and Tech-X Corporation. The original domain of application was laser-plasma interaction, where its results have contributed to leading scientific publications, including a recent article in Nature [6]. Due to the flexibility built into the code, it has grown into many additional areas, like breakdown models in microwave guides or dusty plasmas. The VORPAL framework can currently model the interaction of electromagnetic fields, charged particles, and charged fluids.

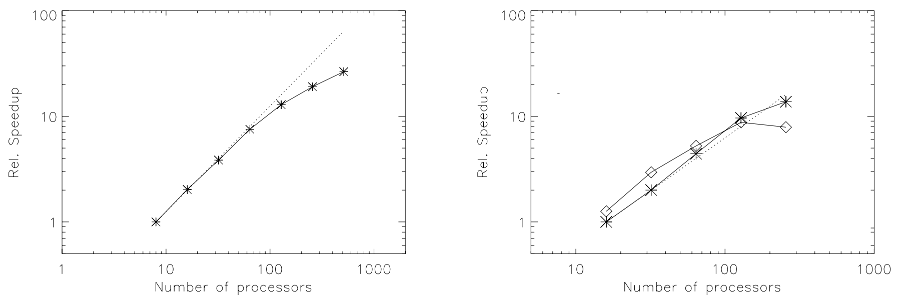


Fig. 1. Speedup figures for the VORPAL code. **Left:** Speedup relative to the execution time on 8 processors for an electromagnetic 3D simulation on a $200 \times 100 \times 100$ cell grid and 5 particles per cell. **Right:** Speedup relative to the execution time on 16 processors for an electrostatic 3D simulation on a $1026 \times 65 \times 65$ cell grid, with different preconditioner to solve Poisson's equation: Gauss-Seidel (Stars) and Algebraic Multigrid (diamonds). In both panels, the perfect speedup is indicated by a dotted line. Speedup figures are taken from References [5] and [7]

The design of VORPAL originates from the understanding that both 2D and 3D simulations will be required in the future: The shorter completion time of 2D simulations allows rapid investigations of physical situations which can then be followed up by more extensive simulations in 3D. The availability of large scale computing facilities, as the 6000 CPU IBM-SP at the National Energy Research Supercomputer Center as well as low-cost Beowulf clusters, affordable to Universities, small corporations and institutes, allows a broad range of users to benefit from this code.

VORPAL uses C++ template meta-programming in order to generate both a 2D and a 3D code from the same code base. It was specifically designed to be a parallel hybrid code, while capable of running as a pure particle or fluid code. All the parallelization aspects of the code are handled transparently. The code uses recursive coordinate bisection, allowing for efficient load balancing even in the case of highly inhomogeneous particle distributions. Passing field, fluid and particle data between processors is efficiently handled, using set theory techniques.

Figure 1 shows the speedup of VORPAL for both a purely electromagnetic PIC simulation and an for an electrostatic problem. In the electrostatic case, Poisson's equation is solved for a potential, using the parallel iterative solvers of the AZTEC library [8].

3 Modeling Dusty Plasmas

An interesting physical question is the temporal evolution of the charge on a dust grain immersed in a two component plasma consisting of electrons and ions. Electrons and ions stream onto the dust grain and 'stick' to it. An uncharged dust grain will be initially charged negatively, due to the higher thermal velocity of the electrons. The resulting strong electric field will then pull ions to the grain, reducing the net charge. In general, analytic solutions are only available for the steady state. This problem is usually treated

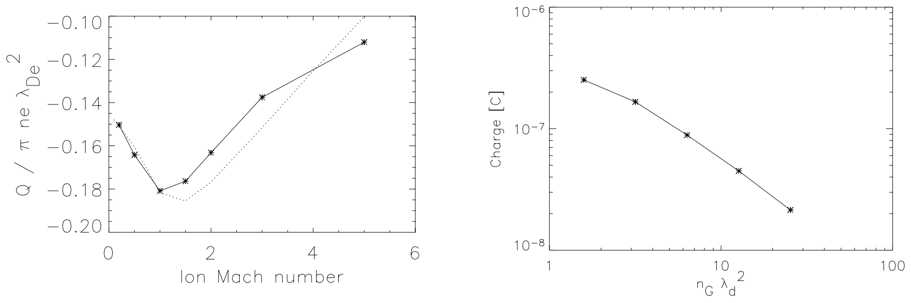


Fig. 2. 2D VORPAL simulations of dusty plasmas. **Left:** Dust charge per particle (stars) as a function of ion flow speed, compared to the theoretical prediction (dotted). **Right:** Average dust charge as a function of dust density

in the electrostatic limit, where the dust grain is modeled as a particle absorbing boundary condition.

However, if the particles velocities become relativistic, the full electromagnetic treatment is desirable. How can a dust grain be modeled in an electromagnetic PIC code? As shown in Sec. 2, the only quantity that is required to update the electromagnetic field self-consistently is the current density, which is inferred from the particle motion. An immobile particle therefore does not contribute to the evolution of the system. If a particle is stopped after it enters a cell effectively leads to an increase of the charge of this cell. An extended, infinitely heavy, perfectly dielectric dust grain can therefore be modeled as a simple particle absorbing area in the computational domain.

4 Preliminary Results

In this section, we present some preliminary results of electromagnetic dust simulations. The dust grains are modeled as stationary, spatially extended cylindrical or spherical particle sinks. The charge-neutral background consists of thermal electrons and protons.

4.1 Asymmetric Charging

Figure 2, left, shows the dependency of the average charge on a single dust grain as a function of the ion flow speed. The simulation was performed in 2D, with a single dust grain in a box of length 2×2 cm on a 100×100 cells grid. The electron temperature is 5 times the ion temperature. The initial electron Debye length is approximately 1 cm. The theoretical results were taken from Lapenta et al.[9], considering the dimensionality of the simulation. The simulations are in good agreement with the theoretical results.

For this simulation, no correction for the depletion of the plasma due to the absorption by the dust grain was performed. This issue will be addressed in further kinetic studies.

4.2 Density Dependent Charging

In a second example, the average charge per dust particle as a function of dust density was investigated. The number of dust grains was varied between 1.5 and 25 per electron

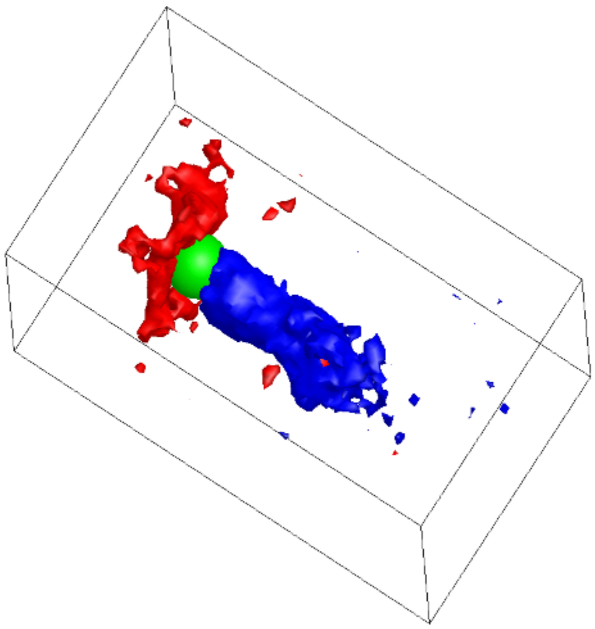


Fig. 3. Dust grain (light gray) in a plasma of streaming ions and thermal electrons. The ions flow from top-left to bottom right. Downstream of the grain, a negatively charged wake forms (dark gray), whereas a positively charged region appears upstream

Debye length. The setup geometry was identical to the previous case, but the electron temperature was chosen to be the same as the ion temperature.

Theory [10] predicts a decrease of the average dust charge for increasing dust density. Figure 2, right, shows this effect reproduced with 2D VORPAL simulations, roughly following a power law. A major difference between our simulations and results published e.g. by Young et al. [11] is that the dust grains in our case are spatially resolved, allowing to investigate additional effects, like wake formation.

4.3 Wake Formation

A third example of our preliminary investigations is wake formation in a system of streaming ions and a thermal electron population. Figure 3 shows the wake of a 3D VORPAL simulation. Clearly visible is the negatively charged tail behind the single dust grain. Also visible is a positively-charged bow in front of the dust grain.

5 Summary

In the previous paragraphs, a model for dust grains in an electromagnetic particle-in-cell code was introduced. It was shown that a particle absorbing region in an electromagnetic PIC code with charge conserving current deposition can act as an infinitely heavy dust grain. This model of a dust grain requires only local information and preserves the parallelism of the electromagnetic PIC algorithm. Good speedup is therefore expected. This will allow to perform studies of spatially extended dust grains in 3D. First simulations of single dust grains immersed in a thermal plasma were presented. The results are in good agreement with previously published results. Future studies will focus on the mutual influence of several dust grains, depending on their relative position. In addition, the time dependent charging of dust grains in 3D will be investigated. All these questions are still unresolved in dusty plasma physics [12].

References

1. Birdsall, Langdon, *Plasma Physics via Computer Simulation*, Adam Hilger, 1991.
2. Hockney, R.W., Eastwood, J.W., *Computer Simulation using Particles*, Adam Hilger, 1988.
3. Villasenor, J., Buneman, O., Comput. Phys. Commun., **69**, 306, 1989.
4. Yee, K.S., Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, IEEE Trans. Antennas Propagat., 14, 302, 1966.
5. Nieter, C. and Cary, J.R., *VORPAL: a versatile plasma simulation code*, J. Comp. Phys, 196 (2), 448, 2004.
6. Geddes, C.G.R, Toth, Cs., van Tilborg, J., Eseray, E., Schroeder, C.B., Bruhwiler, D., Nieter, C., Cary, J.R, *High-quality electron beams from a laser wakefield accelerator using plasma-channel guiding*, Nature, **431**, 538, 2004.
7. Messmer, P., Bruhwiler, D., *An electrostatic solver for the VORPAL code*, Comp. Phys. Comm., 164, 118, 2004.
8. Tuminaro, R.S., Heroux, M., Hutchinson, S.A., Shadid, J.N.,*Official Aztec Users's Guide: Version 2.1*, Technical Report, SAND99-8801J, 1999.
9. Lapenta, G., *Simulation of charging and shielding of dust particles in drifting plasmas*, Phys. o. Plasmas, **6 (5)**, 1442, 1999.
10. Goertz, C.K., *Dusty plasmas in the solar system*, Rev. Geophys, **27 (2)**, 271, 1989.
11. Young, B., Cravens, T.E., Armstrong, T.P., Friauf, R.J., *A two-dimensional particle-in-cell model of a dusty plasma*, J. Geophys. Res., **99**, 2255, 1994.
12. Tsytovich, V.N., Morfill, G.E., Thomas, H., *Complex Plasmas: I. Complex Plasmas as Unusual State of Matter*, Plasma Phys. Reports., **28 (8)**, 625, 2002.

Advanced Algorithms and Software Components for Scientific Computing: An Introduction[⋆]

Organizer: Padma Raghavan

Department of Computer Science and Engineering
The Pennsylvania State University
343K IST Bldg., University Park, PA 16802, USA
raghavan@cse.psu.edu

Abstract. Scientific computing concerns the design, analysis and implementation of algorithms and software to enable computational modeling and simulation across disciplines. Recent research in scientific computing has been spurred both by advances in multiprocessor systems and the wider acceptance of computational modeling as a third mode of scientific investigation, in addition in to theory and experiment. We introduce the next six chapters on advanced algorithms and software components for scientific computing on parallel computers. We review recent results toward enabling the scalable solution of large-scale modeling applications.

1 Introduction

In the last decade, in addition to theory and experiment, computational modeling and simulation has been widely embraced as the third mode of scientific investigation and engineering design. Consequently, the associated computational challenges are being faced by an increasingly larger group of scientists and engineers from a variety of disciplines. This in turn has motivated research in scientific computing aimed at delivering advanced algorithms and software tools that can enable computational science and engineering without specialized knowledge of numerical analysis, computer architectures, and interoperable software components. In this chapter, we introduce some recent research in advanced scientific computing to enable large-scale computational modeling and simulation.

In Section 2, we provide an introduction to the next six chapters covering a range of topics. The first chapter concerns algorithms and software for a scientific database for efficient function approximation in computational fluid dynamics. The next three chapters focus on techniques for scalable, parallel sparse linear system solution. The latter is often the computationally dominant step in simulations of models based on nonlinear partial differential equations using an implicit or semi-implicit scheme with finite-element or finite-difference discretizations. The final two chapters focus on advanced software environments, object-oriented and component frameworks for high-performance scientific software development. We end with concluding remarks in Section 3.

[⋆] This work was supported in part by the National Science Foundation through grants CCF-044345, EIA-0221916, and DMR-0205232.

2 Advanced Algorithms and Software for Scientific Computing

In this section, we provide an introduction to the following six chapters which represent emerging themes of research in scientific computing aimed at meeting the challenges of multi-scale computational modeling on high-performance computing systems.

The first chapter by Plassmann and Veljkovic concerns parallel algorithms for a scientific database for efficient function approximation. The problem is related to the modeling of reacting flows with complex chemistry involving multiple scales, for example, through high-order, Direct Numerical Simulation (DNS). The authors consider the development of a scientific database for approximating computationally expensive functions by archiving previously computed exact values. Earlier, they had demonstrated that sequential algorithms and software for the database problem can be effective in decreasing the running time of complex reacting flow simulations. Now, they consider parallel algorithms and software based on a partitioning of the search space. Their approach involves maintaining a global BSP tree which can be searched on each processor in a manner analogous to the sequential algorithm. They also consider a variety of heuristics for coordinating the distributed management of the database. They establish through experiments that a hybrid of these heuristics exhibits the best performance and scalability by successfully balancing the computational load.

The next three chapters focus on effective parallel sparse linear system solution. Sparse linear solvers can be broadly classified as being 'direct,' 'iterative,' 'domain decomposition,' or 'multilevel/multigrid.' Additionally, there are 'preconditioning' schemes which can be used with iterative solvers to accelerate their convergence. Each class contains a large number of algorithms, and there is multiplicative growth in the number of methods when special forms of multilevel, domain-decomposition, or direct methods are used as preconditioners. The performance of all these methods, including convergence, reliability, execution time, parallel efficiency, and scalability, can vary dramatically depending on the interactions between a specific method, the problem instance, and the computational architecture.

The chapter by D'Ambra, Serafino, and Filippone concerns building effective preconditioners using schemes from domain-decomposition. They focus on constructing parallel Schwarz preconditioners by extending Parallel Sparse BLAS (PSBLAS), a library of routines providing basic linear algebra operations for building iterative sparse linear system solvers on distributed-memory multiprocessors. The next two chapters are related in the sense that they both concern using techniques from sparse direct solvers to develop preconditioners. The chapter by Hénon, Pellegrini, Ramet, Roman and Saad concerns using 'blocked' forms of incomplete factorization to develop robust preconditioners. The chapter by Teranishi and Raghavan concerns parallel preconditioners using incomplete factorizations that are 'latency tolerant.' i.e., preconditioners that can achieve high-efficiency despite the relatively large latency of inter-processor communication on message passing multiprocessors and clusters. All three chapters provide empirical results to demonstrate the efficiency of the preconditioners. These three chapters are representative of an emerging trend in scientific computing where the focus is on hybrid algorithms to provide efficient, scalable and robust sparse solvers that can meet the demands of a range of large-scale applications.

The last two chapters concern software environments for scientific computing. The emphasis is on the design of 'plug-and-play' environments for software development. In such systems, the application developer has access to the wealth of software implementations of a variety of algorithms for different key kernels common to a wide range of simulations. This will enable developers to build software specific to their modeling needs, with relative ease and without sacrificing performance,e by selecting from tuned implementations of advanced algorithms.

The chapter by Heroux and Sala discusses the design of the 'Trilinos' system, a two-level software structure, designed around a collection of 'packages.' Each package focuses on a particular area of research, such as linear and nonlinear solvers or algebraic preconditioners, and is usually developed by a small team of experts in a particular area of research. Packages exist underneath the Trilinos top level, which provides a common look-and-feel to application developers while allowing package interoperability.

The final chapter by Norris concerns software architectures for advanced scientific computing. The Common Component Architecture (CCA) has been developed with the goal of managing the increasing complexity of scientific software development. The CCA attempts to provide a relatively simple component interface requirements with a minimal performance penalty. The chapter explores some concepts and approaches that can contribute to making CCA-based applications easier to design, implement, and maintain. When properly designed, CCA applications can also benefit from hybrid schemes, and adaptive method selection to enhance performance.

3 Conclusions

Recent advances in computing hardware include high-performance multiprocessors like the BlueGene/L from IBM, commodity Beowulf clusters, and their organization into wide-area distributed computational grids. These hold the potential for discovery and design through computational modeling and simulation. However, the raw processing power of the hardware cannot be utilized without the development of sophisticated algorithms and software systems for scientific computing.

Recent decades of research in scientific computing and numerical analysis have resulted in many algorithms for each of the main steps of applications based on models described by nonlinear partial-differential equations (PDEs). The first four of following six chapters concern techniques for developing hybrids of algorithms that were developed earlier in order improve performance or enhance scalability. The last two chapters concern advanced software architectures where such algorithms can be implemented with relative ease within computational modeling and simulation applications.

Although many algorithms have been developed for key computational steps in PDE-based applications, such as sparse linear system solution, and mesh generation and improvement, a growing consensus is that it is neither possible nor practical to predict a priori which algorithm for a given numerical kernel performs best across applications. We therefore anticipate an increased focus on advanced algorithms and software environments that can be used to compose hybrids, and adaptively select methods to improve application performance. The results in the following six chapters reflect this emerging trend.

Extending PSBLAS to Build Parallel Schwarz Preconditioners

Alfredo Buttari[1], Pasqua D'Ambra[2], Daniela di Serafino[3], and Salvatore Filippone[1]

[1] Department of Mechanical Engineering, University of Rome "Tor Vergata"
Viale del Politecnico, I-00133, Rome, Italy
{alfredo.buttari,salvatore.filippone}@uniroma2.it

[2] Institute for High-Performance Computing and Networking, CNR
Via Pietro Castellino 111, I-80131 Naples, Italy
pasqua.dambra@na.icar.cnr.it

[3] Department of Mathematics, Second University of Naples
Via Vivaldi 43, I-81100 Caserta, Italy
daniela.diserafino@unina2.it

Abstract. We describe some extensions to Parallel Sparse BLAS (PSBLAS), a library of routines providing basic Linear Algebra operations needed to build iterative sparse linear system solvers on distributed-memory parallel computers. We focus on the implementation of parallel Additive Schwarz preconditioners, widely used in the solution of linear systems arising from a variety of applications. We report a performance analysis of these PSBLAS-based preconditioners on test cases arising from automotive engine simulations. We also make a comparison with equivalent software from the well-known PETSc library.

1 Introduction

Effective numerical simulations in many application fields, such as Computational Fluid Dynamics, require fast and reliable numerical software to perform Sparse Linear Algebra computations.

The PSBLAS library was designed to provide the kernels needed to build iterative methods for the solution of sparse linear systems on distributed-memory parallel computers [10]. It includes parallel versions of most of the Sparse BLAS *computational kernels* proposed in [7] and a set of *auxiliary routines* for the creation and management of distributed sparse matrix structures. The library provides also *application routines* implementing some sparse Krylov solvers with Jacobi and block Jacobi preconditioners. The library routines have been proved to be powerful and flexible in restructuring a complex CFD code, improving the accuracy and efficiency in the solution of the sparse linear systems and implementing the boundary data exchanges arising in the numerical simulation process [11]. PSBLAS is based on a SPMD programming model and uses the Basic Linear Algebra Communication Subprograms (BLACS) [6] to manage interprocess data exchange. The library is internally written in C and Fortran 77, but provides high-level routine interfaces in Fortran 95. Current development is devoted to extending PSBLAS with preconditioners suitable for fluid flow problems in automotive engine modeling. A basic development guideline is to preserve as much as possible the original PSBLAS software architecture while moving towards the new SBLAS standard [8]. In this work we focus on the extension for building Additive Schwarz preconditioners, widely used in the parallel iterative solution of linear systems arising from PDE discretizations.

The paper is organized as follows. In Section 2 we give a brief description of Additive Schwarz preconditioners. In Section 3 we discuss how the set of PSBLAS data structures and routines has been extended to implement these preconditioners by reusing existing PSBLAS computational routines. In Section 4 we present the results of experiments devoted to analyze the performance of PSBLAS-based Schwarz preconditioners with Krylov solvers on test matrices arising from automotive engine simulations; we also show the results of comparisons with the widely used PETSc library [1]. Finally, in Section 5, we draw some conclusions and future work.

2 An Overview of Additive Schwarz Preconditioners

Schwarz preconditioners are based on domain decomposition ideas originally introduced in the context of variational solution of Partial Differential Equations (see [5,15]). We focus on the algebraic formulation of Additive Schwarz preconditioners for the solution of general sparse linear systems [2,14]; these are widely used with parallel Krylov subspace solvers.

Given the linear system $Ax = b$, where $A = (a_{ij}) \in \Re^{n\times n}$ is a nonsingular sparse matrix with a symmetric non-zero pattern, let $G = (W, E)$ be the adjacency graph of A, where $W = \{1, 2, \ldots, n\}$ and $E = \{(i, j) : a_{ij} \neq 0\}$ are the vertex set and the edge set, respectively. Two vertices are *neighbours* if an edge connects them. A 0-*overlap* partition of W is just an ordinary partition of the graph, i.e. a set of m disjoint nonempty subsets $W_i^0 \subset W$ such that $\cup_{i=1}^m W_i^0 = W$. A δ-*overlap* partition of W with $\delta > 0$ can be defined recursively by considering the sets $W_i^\delta \supset W_i^{\delta-1}$ obtained by including the vertices that are neighbours of the vertices in $W_i^{\delta-1}$. Let n_i^δ be the size of W_i^δ and R_i^δ the $n_i^\delta \times n$ matrix formed by the row vectors e_j^T of the $n \times n$ identity matrix, with $j \in W_i^\delta$. For each $v \in \Re^n$, $R_i^\delta v$ is the vector containing the components of v corresponding to the vertices in W_i^δ, hence R_i^δ can be viewed as a restriction operator from $\Re^n$ to $\Re^{n_i^\delta}$. Likewise, the transpose matrix $(R_i^\delta)^T$ can be viewed as a prolongation operator from $\Re^{n_i^\delta}$ to $\Re^n$. The *Additive Schwarz (AS)* preconditioner, M_{AS}, is then defined by

$$M_{AS}^{-1} = \sum_{i=1}^{m} (R_i^\delta)^T (A_i^\delta)^{-1} R_i^\delta,$$

where the $n_i^\delta \times n_i^\delta$ matrix $A_i^\delta = R_i^\delta A (R_i^\delta)^T$ is obtained by considering the rows and columns of A corresponding to the vertices in W_i^δ.

When $\delta = 0$ M_{AS} is the well-known block Jacobi preconditioner. The convergence theory for the AS preconditioner is well developed in the case of symmetric positive definite matrices (see [5,15] and references therein). Roughly speaking, when the AS preconditioner is used in conjunction with a Krylov subspace method, the convergence rapidly improves as the overlap δ increases, while it deteriorates as the number m of subsets W_i^δ grows. Theoretical results are available also in the case of nonsymmetric and indefinite problems (see [4,15]).

Recently some variants of the AS preconditioner have been developed that outperform the classical AS for a large class of matrices, in terms of both convergence rate and of computation and communication time on parallel distributed-memory computers [3,9,12]. In particular, the *Restricted Additive Schwarz (RAS)* preconditioner, M_{RAS}, and the *Additive Schwarz* preconditioner *with Harmonic extension (ASH)*, M_{ASH}, are defined by

$$M_{RAS}^{-1} = \sum_{i=1}^{m} (\tilde{R}_i^0)^T (A_i^\delta)^{-1} R_i^\delta, \qquad M_{ASH}^{-1} = \sum_{i=1}^{m} (R_i^\delta)^T (A_i^\delta)^{-1} \tilde{R}_i^0,$$

where $\tilde{R}_i^0$ is the $n_i^\delta \times n$ matrix obtained by zeroing the rows of R_i^δ corresponding to the vertices in $W_i^\delta \backslash W_i^0$. The application of the AS preconditioner requires the solution of m independent linear systems of the form

$$A_i^\delta w_i = R_i^\delta v \tag{1}$$

and then the computation of the sum

$$\sum_{i=1}^{m} (R_i^\delta)^T w_i. \tag{2}$$

In the RAS preconditioner R_i^δ in (2) is replaced by $\tilde{R}_i^0$; hence, in a parallel implementation where each processor holds the rows of A with indices in W_i^0 and the corresponding components of right-hand side and solution vectors, this sum does not require any communication. Analogously, in the ASH preconditioner R_i^δ in equation (1) is replaced by $\tilde{R}_i^0$; therefore, the computation of the right-hand side does not involve any data exchange among processors.

In the applications, the exact solution of system (1) is often prohibitively expensive. Thus, it is customary to substitute the matrix $(A_i^\delta)^{-1}$ with an approximation $(K_i^\delta)^{-1}$, computed by incomplete factorizations, such as ILU, or by iterative methods, such as SSOR or Multigrid (see [5]).

3 Building and Applying AS Preconditioners in PSBLAS

We now review the basic operations involved in the Additive Schwarz preconditioners from the point of view of parallel implementation through PSBLAS routines. We begin by drawing a distinction between

Preconditioner Setup: the set of basic operations needed to identify W_i^δ, to build A_i^δ from A, and to compute K_i^δ from A_i^δ;

Preconditioner Application: the set of basic operations needed to apply the restriction operator R_i^δ to a given vector v, to compute (an approximation of) w_i, by applying $(K_i^\delta)^{-1}$ to the restricted vector, and, finally, to obtain sum (2).

Before showing how PSBLAS has been extended to implement the Additive Schwarz preconditioners described in Section 2, we briefly describe the main data structures involved in the PSBLAS library. A general row-block distribution of sparse matrices is supported by PSBLAS, with conformal distribution of dense vectors.

Sparse Matrix: Fortran 95 derived data type, called `D_SPMAT`, it includes all the information about the local part of a distributed sparse matrix and its storage mode, following the Fortran 95 implementation of a sparse matrix in the sparse BLAS standard of [7].

Communication Descriptor: Fortran 95 derived data type `DESC_TYPE`; it contains a representation of the sets of indices involved in the parallel data distribution, including the 1-overlap indices, i.e. the set $W_i^1 \backslash W_i^0$, that is preprocessed for the data exchange needed in sparse matrix-vector products.

Existing PSBLAS computational routines implement the operations needed for the application phase of AS preconditioners, provided that a representation of the δ-partition is built and packaged into a new suitable data structure during the phase of preconditioner setup. The next two sections are devoted to these issues.

3.1 PSBLAS Implementation of Preconditioner Application

To compute the right-hand side in (1) the restriction operator R_i^δ must be applied to a vector v distributed among parallel processes conformally to the sparse matrix A. On each process the action of R_i^δ corresponds to gathering the entries of v with indices belonging to the set $W_i^\delta \backslash W_i^0$. This is the semantics of the PSBLAS `F90_PSHALO` routine, which updates the halo components of a vector, i.e. the components corresponding to the 1-overlap indices. The same code can apply an arbitrary δ-overlap operator, if a suitable auxiliary descriptor data structure is provided.

Likewise, the computation of sum (2) can be implemented through a suitable call to the PSBLAS computational routine `F90_PSOVRL`; this routine can compute either the sum, the average or the square root of the average of the vector entries that are replicated in different processes according to an appropriate descriptor.

Finally, the computation of $(K_i^\delta)^{-1} v_i^\delta$, where $v_i^\delta = R_i^\delta v$ or $v_i^\delta = \tilde{R}_i^0 v$, can be accomplished by two calls to the sparse block triangular solve routine `F90_PSSPSM`, given a local (incomplete) factorization of A_i^δ.

Therefore, the functionalities needed to implement the application phase of the AS, RAS and ASH preconditioners, in the routine `F90_ASMAPPLY`, are provided by existing computational PSBLAS routines, if an auxiliary descriptor `DESC_DATA` is built. Thus, the main effort in implementing the preconditioners lies in the definition of a preconditioner data structure and of routines for the preconditioner setup phase, as discussed in Section 3.2.

3.2 PSBLAS Implementation of Preconditioner Setup

To implement the application of AS preconditioners we defined a data structure `PREC_DATA` that includes in a single entity all the items involved in the application of the preconditioner:

```
TYPE PREC_DATA
INTEGER                      :: PREC
INTEGER                      :: N_OVR
TYPE(D_SPMAT)                :: L, U
REAL(KIND(1.D0)), POINTER :: D(:)
TYPE(DESC_TYPE)     :: DESC_DATA
END TYPE PREC_DATA
```

Fig. 1. Preconditioner data type

- a preconditioner identifier, PREC, and the number δ of overlap layers, N_OVR;
- two sparse matrices L and U, holding the lower and upper triangular factors of K_i^δ (the diagonal of the upper factor is stored in a separate array, D);
- the auxiliary descriptor DESC_DATA, built from the sparse matrix A, according to N_OVR.

This data structure has the Fortran 95 definition shown in Figure 1. Note that the sparse matrix descriptor is kept separate from the preconditioner data; with this choice the sparse matrix operations needed to implement a Krylov solver are independent of the choice of the preconditioner.

The algorithm to setup an instance P of the PREC_DATA structure for AS, RAS or ASH, with overlap N_OVR, is outlined in Figure 2. By definition the submatrices A_i^0 identify the vertices in W_i^1; the relevant indices are stored into the initial communication descriptor. Given the set W_i^1, we may request the column indices of the non-zero entries in the rows corresponding to $W_i^1 \backslash W_i^0$; these in turn identify the set W_i^2, and so on. All the communication is performed in the steps 4.2 and 6, while the other steps are performed locally by each process. A new auxiliary routine, F90_DCSRSETUP, has been developed to execute the steps 1–6. To compute the triangular factors of K_i^δ (step 7), the existing PSBLAS routine F90_DCSRLU, performing an ILU(0) factorization of A_i^δ, is currently used. The two previous routines have been wrapped into a single PSBLAS application routine, named F90_ASMBUILD.

It would be possible to build the matrices A_i^δ while building the auxiliary descriptor DESC_DATA. Instead, we separated the two phases, thus providing the capability to reuse the DESC_DATA component of an already computed preconditioner; this allows efficient handling of common application situations where we have to solve multiple linear systems with the same structure.

4 Numerical Experiments

The PSBLAS-based Additive Schwarz preconditioners, coupled with a BiCGSTAB Krylov solver built on the top of PSBLAS, were tested on a set of matrices from automotive engine simulations.

These matrices arise from the pressure correction equation in the implicit phase of a semi-implicit algorithm (ICED-ALE [13]) for the solution of unsteady compressible Navier-Stokes equations, implemented in the KIVA application software, as modified

```
1. Initialize the descriptor P%PREC_DATA by copying the matrix descriptor DESC_A.
2. Initialize the overlap level: $\eta = 0$.
3. Initialize the local vertex set, $W_i^\eta = W_i^0$, based on the current descriptor.
4. While ( $\eta <$ N_OVR )
   4.1 Increase the overlap: $\eta = \eta + 1$.
   4.2 Build $W_i^\eta$ from $W_i^{\eta-1}$, by adding the halo indices of $A_i^{\eta-1}$, and exchange
       with other processors the column indices of the non-zero entries in the rows
       corresponding to $W_i^\eta \backslash W_i^{\eta-1}$.
   4.3 Compute the halo indices of $A_i^\eta$ and store them into P%DESC_DATA.
   Endwhile
5. If ( N_OVR $> 0$ ) Optimize the descriptor P%DESC_DATA and store it in its final
   format.
6. Build the enlarged matrix $A_i^\delta$, by exchanging rows with other processors.
7. Compute the triangular factors of the approximation $K_i^\delta$ of $A_i^\delta$.
```

Fig. 2. Preconditioner setup algorithm

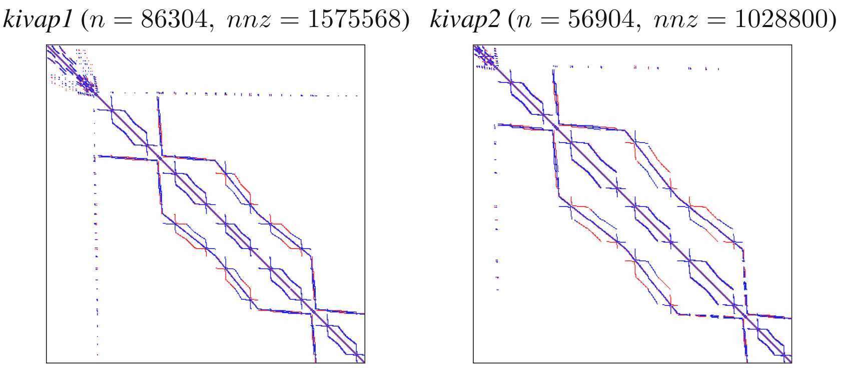


Fig. 3. Sparsity patterns of the test matrices

in [11]. The test case is a simulation of a commercial direct injection diesel engine featuring a bowl shaped piston, with a mesh containing approximately 100000 control volumes; during the simulation mesh layers are activated/deactivated following the piston movement. We show measurements for two matrices, *kivap1* and *kivap2*, corresponding to two different simulation stages. They have sizes 86304 and 56904, respectively, with symmetric sparsity patterns and up to 19 non-zeroes per row (see Figure 3).

Numerical experiments were carried out on a Linux cluster, with 16 PCs connected via a Fast Ethernet switch, at the Department of Mathematics of the Second University of Naples. Each PC has a 600 MHz Pentium III processor, a memory of 256 MB and an L2 cache of 256 KB. PSBLAS was installed on the top of the BLAS reference implementation, BLACS 1.1 and mpich 1.2.5, using the gcc C compiler, version 2.96, and the Intel Fortran compiler, version 7.1.

All the tests were performed using a row-block distribution of the matrices. Right-hand side and solution vectors were distibuted conformally. The number m of vertex sets

Table 1. Iteration counts of RAS-preconditioned BiCGSTAB

np	*kivap1*								*kivap2*							
	PSBLAS				*PETSc*				*PSBLAS*				*PETSc*			
	δ=0	δ=1	δ=2	δ=4	δ=0	δ=1	δ=2	δ=4	δ=0	δ=1	δ=2	δ=4	δ=0	δ=1	δ=2	δ=4
1	12	12	12	12	12	12	12	12	33	33	33	33	33	33	33	33
2	16	13	11	12	16	12	12	12	50	34	33	34	46	34	32	32
4	16	13	12	13	16	12	11	12	47	36	34	34	50	35	32	31
6	18	13	13	12	18	12	12	12	48	37	38	34	52	35	34	29
8	20	14	14	12	20	12	13	12	50	37	37	35	53	36	32	32
10	19	13	12	13	19	13	12	12	54	38	35	34	51	33	32	33
12	21	14	12	12	21	13	13	12	54	34	38	33	53	35	32	33
14	20	14	13	14	20	12	13	12	51	36	38	33	52	35	32	32
16	22	15	14	13	22	13	12	12	58	37	38	36	59	38	32	30

W_i^δ used by the preconditioner was set equal to the number of processors; the right-hand side and the initial approximation of the solution of each linear system were set equal to the unit vector and the null vector, respectively. The preconditioners were applied as right preconditioners. The BiCGSTAB iterations were stopped when the 2-norm of the residual was reduced by a factor of 10^{-6} with respect to the initial residual, with a maximum of 500 iterations.

We also compared the PSBLAS implementation of the RAS-preconditioned BiCGSTAB solver with the corresponding implementation provided by the well-known PETSc library [1], on the same test problems and with the same stopping criterion. The experiments were performed using PETSc 2.1.6, compiled with gcc 2.96 and installed on the top of mpich 1.2.5 and of the BLAS and LAPACK implementations provided with the Linux Red Hat 7.1 distribution.

We report performance results with RAS; this was in general the most effective Additive Schwarz variant, in accordance with the literature cited in Section 2. In Table 1 we show the iteration counts of RAS-preconditioned BiCGSTAB for PSBLAS and PETSc on both test problems, varying the number np of processors and the overlap δ. We see that the number of iterations significantly decreases in passing from a 0-overlap to a 1-overlap partition, especially for *kivap2*, while it does not have relevant changes with a further growth of the overlap. We note also that, for $\delta > 0$, the number of iterations is almost stable as the number of processes increases. This behaviour appears to be in agreement with the sparsity pattern of the test matrices. The number of iterations of PSBLAS is generally comparable with the one of PETSc. For *kivap2*, in a few cases a difference of 5 or 6 iterations is observed, which is due to some differences in the row-block distribution of the matrix and in the row ordering of the enlarged matrices A_i^δ used by the two solvers.

In Figure 4 we show the execution times of the RAS-preconditioned BiCGSTAB on the selected test cases, varying the number of processors. These times include also the preconditioner setup times. As expected, the time usually decreases as the number of processors increases; a slight time increase can be observed in a few cases, which can be mainly attributed to the row-block distribution of the matrices. A more regular

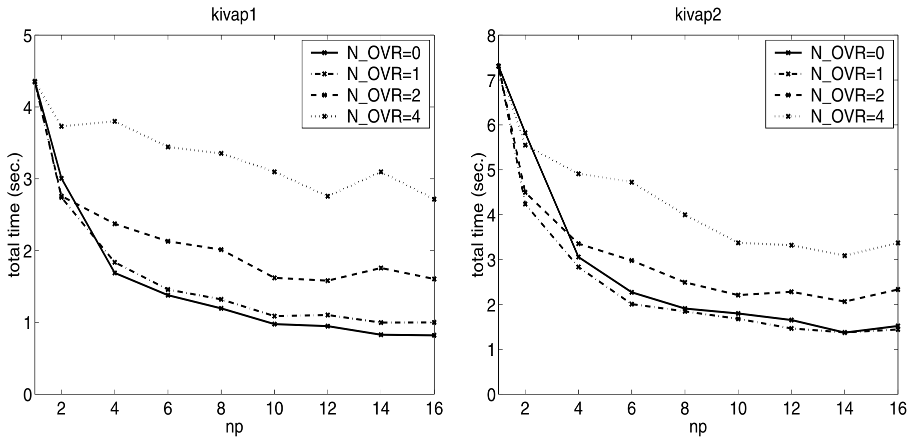


Fig. 4. Execution times of PSBLAS RAS-preconditioned BiCGSTAB

behaviour is expected by applying suitable graph partitioning algorithms to the initial data distribution. For *kivap1* the times are generally lower for overlap 0. Indeed, the small decrease in the number of BiCGSTAB iterations, as the overlap grows, is not able to balance the cost of the preconditioner setup phase. For *kivap2*, the reduction of the number of iterations obtained with overlap 1 is able to compensate the setup cost, thus leading to the smallest execution times.

Finally, in Figure 5 we compare the execution times of PSBLAS and PETSc. The performance of the two solvers is generally comparable. PSBLAS is always faster on a small number of processors, whereas for higher levels of overlap (2 and 4) PETSc requires less execution time as the number of processors increases. A more detailed analysis has shown that this behaviour is due to the smaller preconditioner setup time of PETSc. This issue is currently under investigation, taking into account the different choices implemented by PSBLAS and PETSc in the setup of the preconditioners.

5 Conclusions and Future Work

We presented some results of an ongoing activity devoted to updating and extending PSBLAS, a parallel library providing basic Linear Algebra operations needed to build iterative sparse linear system solvers on distributed-memory parallel computers. Motivations for our work come from the flexibility and effectiveness shown by PSBLAS in restructuring and parallelizing a legacy CFD code [11], still widely used in the automotive engine application world, and also from the appearance of a new proposal for the serial Sparse BLAS standard [8].

In this paper we focused on the extension of PSBLAS to implement different versions of Additive Schwarz preconditioners. On test problems from automotive engine simulations the preconditioners showed performances comparable with those of other well-established software. We are currently working on design and implementation issues concerning the addition of a coarse-grid solution step to the basic Additive Schwarz preconditioners, in order to build a two-level Schwarz preconditioning module.

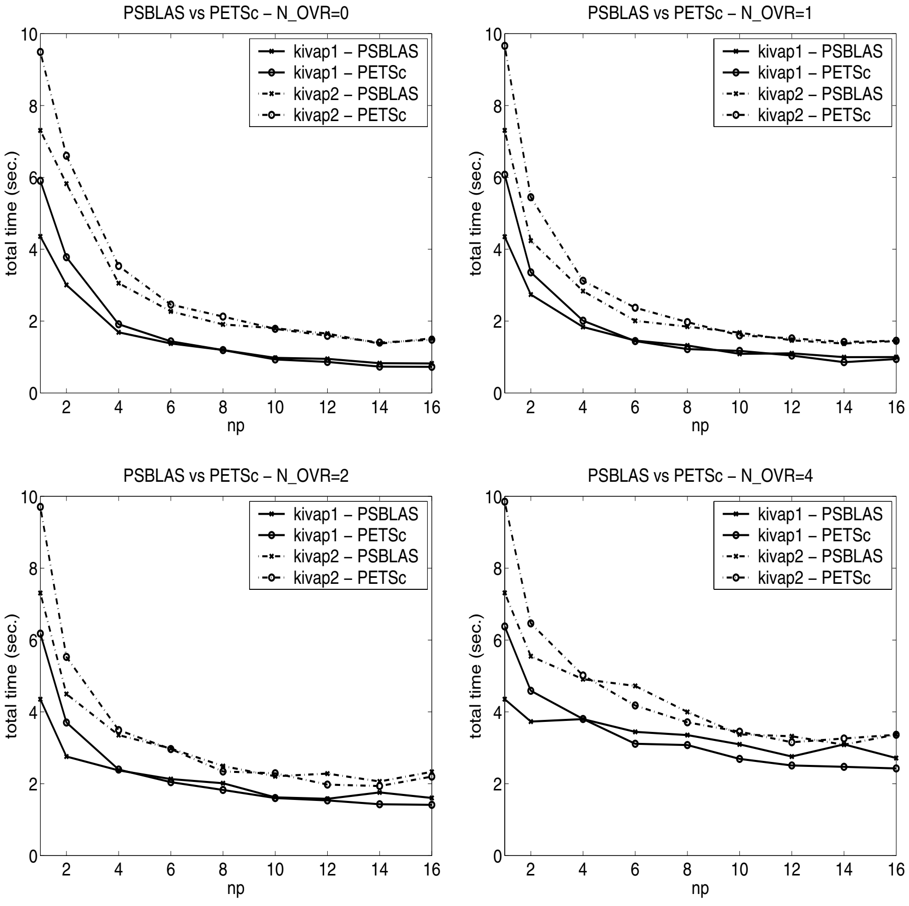


Fig. 5. Comparison of execution times of the PSBLAS and PETSc

References

1. S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. Knepley, L. Curfman McInnes, B. F. Smith, H. Zhang. PETSc Users Manual. Tech. Rep. ANL-95/11, Revision 2.1.6, Argonne National Laboratory, August 2003.
2. X. C. Cai, Y. Saad. Overlapping Domain Decomposition Algorithms for General Sparse Matrices. *Num. Lin. Alg. with Applics.*, 3(3):221–237, 1996.
3. X.C. Cai, M. Sarkis. A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems. *SIAM J. Sci. Comput.*, 21(2):792–797, 1999.
4. X.C. Cai, O. B. Widlund. Domain Decomposition Algorithms for Indefinite Elliptic Problems. *SIAM J. Sci. Stat. Comput.*, 13(1):243–258, 1992.
5. T. Chan, T. Mathew. Domain Decomposition Algorithms. In A. Iserles, editor, *Acta Numerica 1994*, pages 61–143, 1994. Cambridge University Press.
6. J. J. Dongarra, R. C. Whaley. A User's Guide to the BLACS v. 1.1. Lapack Working Note 94, Tech. Rep. UT-CS-95-281, University of Tennessee, March 1995 (updated May 1997).

7. I. Duff, M. Marrone, G. Radicati, C. Vittoli. Level 3 Basic Linear Algebra Subprograms for Sparse Matrices: a User Level Interface. *ACM Trans. Math. Softw.*, 23(3):379–401, 1997.
8. I. Duff, M. Heroux, R. Pozo. An Overview of the Sparse Basic Linear Algebra Subprograms: the New Standard from the BLAS Technical Forum. *ACM Trans. Math. Softw.*, 28(2):239–267, 2002.
9. E. Efstathiou, J. G. Gander. Why Restricted Additive Schwarz Converges Faster than Additive Schwarz. *BIT Numerical Mathematics*, 43:945–959, 2003.
10. S. Filippone, M. Colajanni. PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices. *ACM Trans. Math. Softw.*, 26(4):527–550, 2000.
11. S. Filippone, P. D'Ambra, M. Colajanni. Using a Parallel Library of Sparse Linear Algebra in a Fluid Dynamics Applications Code on Linux Clusters. In G. Joubert, A. Murli, F. Peters, M. Vanneschi, editors, *Parallel Computing - Advances & Current Issues*, pages 441–448, 2002. Imperial College Press.
12. A. Frommer, D. B. Szyld. An Algebraic Convergence Theory for Restricted Additive Schwarz Methods Using Weighted Max Norms. *SIAM J. Num. Anal.*, 39(2):463–479, 2001.
13. C. W. Hirt, A. A. Amsden, J. L. Cook. An Arbitrary Lagrangian-Eulerian Computing Method for All Flow Speeds. *J. Comp. Phys.*, 14:227–253, 1974.
14. Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd edition, 2003.
15. B. Smith, P. Bjorstad, W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

A Direct Orthogonal Sparse Static Methodology for a Finite Continuation Hybrid LP Solver

Pablo Guerrero-García and Ángel Santos-Palomo

Department of Applied Mathematics, Complejo Tecnológico
University of Málaga, Campus de Teatinos s/n
29071 Málaga, Spain
{pablito,santos}@ctima.uma.es

Abstract. A finite continuation method for solving linear programs (LPs) has been recently put forward by K. Madsen and H. B. Nielsen which, to improve its performance, can be thought of as a Phase-I for a non-simplex active-set method (also known as basis-deficiency-allowing simplex variation); this avoids having to start the simplex method from a highly degenerate square basis. An efficient sparse implementation of this combined hybrid approach to solve LPs requires the use of the same sparse data structure in both phases, and a way to proceed in Phase-II when a non-square working matrix is obtained after Phase-I.
In this paper a direct sparse orthogonalization methodology based on Givens rotations and a static sparsity data structure is proposed for both phases, with a LINPACK-like downdating without resorting to hyperbolic rotations. Its sparse implementation (recently put forward by us) is of reduced-gradient type, regularization is not used in Phase-II, and occasional refactorizations can take advantage of row orderings and parallelizability issues to decrease the computational effort.

Keywords: linear programming, sparse direct methods, orthogonalization, Givens rotations, static sparsity structure, sparse update & downdate, non-simplex active-set methods, basis-deficiency-allowing simplex variations, Phase-I, finite continuation hybrid method, reduced-gradient, non-square working matrix.

1 Introduction

In this paper a two-phase hybrid method for solving linear programs (LPs) like that described by Nielsen in [16, §5.5] is considered, the main difference being that a non-simplex active-set method [18] (also known as a basis-deficiency-allowing simplex variation, cf. [17]) is used in Phase-II instead of the simplex method to avoid having to start it from a highly degenerate square basis. This proposed linear programming (LP) framework is briefly described in §2.

The main goal of the paper is to develop an useful sparse implementation methodology which could be taken as the stepping stone to a robust sparse implementation of this LP method which could take advantage of parallel architectures when refactorizations of the working matrix have to be done, and the latter sections are devoted to describe the sparse orthogonal factorizations used to accomplish such goal. Hence a static data structure come into play, and after describing in §3 the original technique used in Phase-II, the modification of this technique to be used also in Phase-I is given in §4. The key point is that the same sparse static data structure is then used in all phases. Additional features are included in §5.

The notation used here is as follows. Let us consider the usual unsymmetric primal-dual pair of LPs using a non-standard notation (we have deliberately exchanged the usual roles of b and c, x and y, n and m, and (P) and (D), in a clear resemblance with the work of Madsen, Nielsen and Pinar [8, p. 614]):

$$
\begin{array}{llll}
(P) & \min\ \ell(x) \doteq c^T x\ ,\ x \in \mathbb{R}^n & \qquad (D) & \max\ \mathcal{L}(y) \doteq b^T y\ ,\ y \in \mathbb{R}^m \\
 & \text{s.t}\ \ A^T x \geq b & & \text{s.t}\ \ Ay = c \qquad ,\ y \geq \mathrm{O}
\end{array}
$$

where $A \in \mathbb{R}^{n\times m}$ with $m \geq n$ and $\operatorname{rank}(A) = n$. We denote with $\mathcal{F}$ and $\mathcal{G}$ the feasible regions of (P) and (D), respectively, and with A_k the current working set matrix formed by a subset of the columns of the fixed matrix A. For example, the sparse structure of the A matrix of the AFIRO problem from NETLIB is shown in the left part of figure 1, with $n = 27$ and $m = 51$; moreover, the working set matrix A_{25} corresponding to the working set $\mathcal{B}^{(25)}$ in iteration 25 could be

$$A_{25} \doteq A(:\,, \mathcal{B}^{(25)}) = A(:\,, [9\ 16\ 18\ 22\ 29\ 40\ 43\ 44\ 50\ 51]) \qquad (1)$$

where MATLAB notation has been used. We shall indicate with m_k the number of elements of the ordered basic set $\mathcal{B}^{(k)}$, with $\mathcal{N}^{(k)} \doteq [1\colon m]\backslash\mathcal{B}^{(k)}$ and with $A_k \in \mathbb{R}^{n\times m_k}$ and $N_k \in \mathbb{R}^{n\times(m-m_k)}$ the submatrices of A formed with the columns included in $\mathcal{B}^{(k)}$ and $\mathcal{N}^{(k)}$, respectively, and $m_k \leq n$ and $\operatorname{rank}(A_k) = m_k$ are only required in Phase-II; as usual, the ith column of A is denoted by a_i, and consistently the matrix whose ith row is a_i^T is denoted by A^T. We shall indicate with $Z_k \in \mathbb{R}^{n\times(n-m_k)}$ a matrix whose columns form a basis (not necessarily orthonormal) of the null space of A_k^T. Superindex $^{(k)}$ will be withheld when it is clear from context.

2 Proposed LP Framework

A two-phase hybrid method like that described by Nielsen in [16, §5.5] is considered, where the Phase-I is based on computing —by Newton method— a minimizer of a piecewise quadratic, see [15]. The dual feasibility is maintained during his Phase-I, as pointed

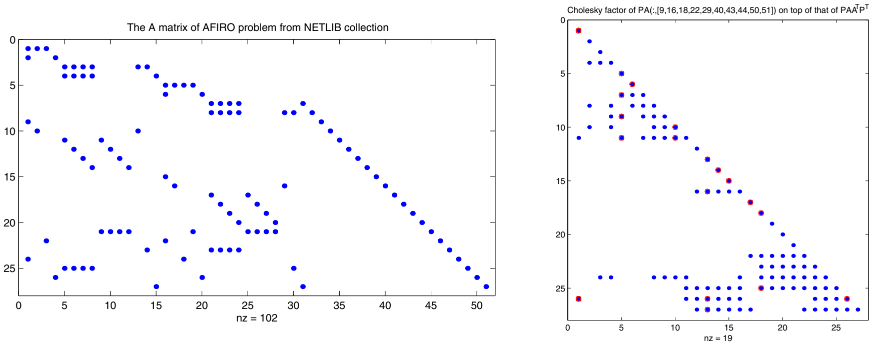


Fig. 1. Sparse structures

out by Nielsen [16, p. 90]; then a crossover to obtain a vertex of $\mathcal{G}$ is accomplished, and finally a non-simplex active-set method is employed to obtain primal feasibility (and hence optimality, as dual feasibility is maintained through both the crossover and the Phase-II).

Nielsen [16, pp. 97–98] has shown that all we have to do during this Phase-I is to device an efficient way to deal with a sequence of sparse linear system of equations whose coefficient matrix is $\sigma I_n + A_k A_k^T$ (constant σ), which is positive definite. Once an $y \in \mathcal{G}$ is obtained, a vertex of $\mathcal{G}$ has to be obtained for the Phase-II to start with. We resort to the well-known constructive result about the existence of a basic feasible solution for a system of linear equality constraints in nonnegative variables (cf. [12, §3.5.4], [13, §11.4.1.5], [9, §2.1.1]), which is closely related with the first stage of the celebrated crossover procedure of Megiddo [11]. Instead of a Gaussian-like methodology, the constraints determined by the Phase-I are firstly rotated one after another in the sparse static structure described in §3, and a dual ascent direction is then computed by projecting the appropriate subvector of b onto the null space of the current working set. A deletion or an exchange is performed accordingly in the working set, as well as a systematic updating of $y \in \mathcal{G}$ after the usual primal min-ratio.

The crucial point to be shown here is that the final (full-column rank) working matrix after the completion of this crossover is not restricted to be square. As an illustration, let us consider the following crossover examples:

Example 1 [12, p. 122]: $y^{(0)} = [1; 2; 3; 4; 0; 0; 0]$,

$$\left[\begin{array}{c|c} b^T & \\ \hline A & c \end{array}\right] = \left[\begin{array}{ccccccc|c} 10 & -4 & -6 & -2 & -4 & -8 & -10 & \\ \hline 1 & 1 & 3 & 4 & 0 & 0 & 0 & 28 \\ 0 & -1 & -1 & -2 & 1 & 0 & 0 & -13 \\ 0 & 1 & 1 & 2 & 0 & 1 & 0 & 13 \\ 1 & 0 & 2 & 2 & 0 & 0 & -1 & 15 \end{array}\right],$$

k	$\mathcal{B}^{(k)}$	$\mathcal{L}^{(k)}$
0	$\{1,2,3,4\}$	−24.0
1	$\{1,2,4\}$	+10.9
2	$\{1,2\}$	+98.0

Note that $n = 4$ but Phase-II has to be started with $m_2 = 2$, hence if we would have to proceed with the simplex method as in [16, §5.5], then a highly degenerate square basis would arise. This is the reason why we prefer a basis-deficiency-allowing simplex variation like that described at the end of this section.

Example 2 [13, pp. 475–476]: $y^{(0)} = [5; 12; 13; 1; 2; 0; 0]/2$,

$$\left[\begin{array}{c|c} b^T & \\ \hline A & c \end{array}\right] = \left[\begin{array}{ccccccc|c} 10 & -4 & -6 & -2 & -4 & -8 & -10 & \\ \hline 1 & 0 & 0 & 1 & 0 & 1 & -1 & 3 \\ 0 & 1 & 0 & 0 & -1 & 2 & -1 & 5 \\ 0 & 0 & 1 & -1 & 1 & 1 & -2 & 7 \end{array}\right],$$

k	$\mathcal{B}^{(k)}$	$\mathcal{L}^{(k)}$
0	$\{1,2,3,4,5\}$	−43.0
1	$\{1,2,3,5\}$	−33.3
2	$\{1,2,3\}$	−32.0

In this case $n = 3$ and $m_2 = 3$, so we proceed with the usual simplex method, although we want to use the same sparse data structure as that used in Phase-I.

Example 3 [12, p. 123, Ex. 3.24a]: $y^{(0)} = [1;2;3;4;5;0;0]$,

$$\left[\begin{array}{c|c} b^T & \\ \hline A & c \end{array}\right] = \left[\begin{array}{ccccccc|c} -1 & -1 & -1 & -2 & -2 & -2 & -4 & \\ \hline 1 & 1 & 0 & 2 & 1 & 0 & 1 & 16 \\ 0 & 1 & 1 & 1 & 2 & 0 & 5 & 19 \\ 1 & 0 & 1 & 1 & 1 & 2 & 2 & 13 \end{array}\right],$$

k	$\mathcal{B}^{(k)}$	$\mathcal{L}^{(k)}$
0	$\{1,2,3,4,5\}$	-24.0

In this case $n = 3$ and $m_0 = 5$, but we do not have to proceed to Phase-II because optimality has been detected during the crossover.

Example 4 [12, p. 123, Ex. 3.24b]: $y^{(0)} = [1;2;3;4;5;0;0]$,

$$\left[\begin{array}{c|c} b^T & \\ \hline A & c \end{array}\right] = \left[\begin{array}{ccccccc|c} -1 & -1 & -1 & -2 & -2 & 0 & -4 & \\ \hline 1 & 1 & 0 & 2 & 1 & 0 & 1 & 16 \\ 0 & 1 & 1 & 1 & 2 & 0 & 5 & 19 \\ 1 & 0 & 1 & 1 & 1 & 2 & 2 & 13 \end{array}\right],$$

k	$\mathcal{B}^{(k)}$	$\mathcal{L}^{(k)}$
0	$\{1,2,3,4,5\}$	-24.0
1	$\{2,3,4,5,6\}$	-23.2
2	$\{2,4,5,6\}$	-22.4
3	$\{2,5,6\}$	-19.0

The goal of this example is twofold, because it illustrates the facts that an exchange can take place during the crossover, and that the final working set obtained after the crossover is not necessarily a subset of that obtained after the Phase-I.

Finally, the non-simplex active-set method (cf. Santos-Palomo and Guerrero-García [18]) is a primal-feasibility search loop in which a violated constraint is chosen to enter the working set, and a constraint is selected for leaving the working set only when the normal of this entering constraint is in the range space of the current working matrix. Note that an addition or an exchange is thus done in each iteration, whereas an exchange would always be done in the simplex method. A preliminary check of the independence of the constraint to be rotated next is performed to decide whether an addition or an exchange takes place. We omit here the scheme of this Phase-II, and refer the interested reader to [18] for additional details.

3 Original Technique Used in Phase-II

The direct orthogonal sparse methodology that we are about to describe dispenses with an explicit orthogonal (dense) factor because of sparsity considerations. It works on top of a sparse triangular matrix, namely the static structure of the Cholesky factor R of AA^T, and a permutation $P \in \mathbb{R}^{n\times n}$ of the rows of A takes place *a priori* to improve the sparsity of R. Denoting with R_k a Cholesky factor of $A_kA_k^T$, a well-known result [4] from sparse linear algebra is that $R_k \subset R$ in a structural sense, namely that we have enough memory already allocated in the static structure which has been *a priori* set up whatever A_k (and consequently R_k) is.

As an illustration, the sparse structure of a (transposed) Cholesky factor R_{25}^T of $PA_{25}A_{25}^TP^T$ in (1) is shown in the right part of figure 1 (big dots) on top of that of PAA^TP^T (small dots). Note that when all the small dots are set to zero, a column-echelon permuted lower trapezoidal matrix would arise if null columns were deleted; the transpose of this matrix is what we have to maintain.

Bearing the static structure previously described in mind, let us show how the implementation of the Phase-II is accomplished with reduced-gradient type techniques described by Santos-Palomo and Guerrero-García [19], where the sparse QR factorization of A_k^T is used to maintain a sparse Cholesky factor R_k of $A_k A_k^T$, with A_k now being a full column rank matrix. In particular, there exists an *implicitly defined* (by the staircase shape of the structure) permutation Π_k^T of the columns of A_k^T such that

$$A_k^T \Pi_k^T = Q_k R_k \doteq Q_k \left[R_\text{i} \; R_\text{d} \right], \quad Q_k, R_\text{i} \in \mathbb{R}^{m_k \times m_k} \text{ and } R_\text{d} \in \mathbb{R}^{m_k \times (n-m_k)},$$

with R_k upper trapezoidal, R_i upper triangular and nonsingular, and Q_k orthogonal. Nevertheless, the permutation Π_k^T is not explicitly performed and Q_k is avoided due to sparsity considerations, thus the row-echelon permuted upper trapezoidal matrix $R_k \Pi_k$ is what we have to maintain. As an example, $\Pi_k^T = [e_1, e_3, e_6, e_2, e_4, e_5, e_7]$ in

$$\underbrace{\texttt{qr}\left(\begin{bmatrix} \text{X X X X X X X} \\ \text{X X X X X X X} \\ \text{X X X X X X X} \end{bmatrix}\right)}_{A_k^T} = \underbrace{\begin{bmatrix} \text{X X X} \\ \text{X X X} \\ \text{X X X} \end{bmatrix}}_{Q_k} \cdot \underbrace{\begin{bmatrix} \text{X X X X X X X} \\ \text{O O X X X X X} \\ \text{O O O O O X X} \end{bmatrix}}_{R_k \cdot \Pi_k}$$

Note that a reordering of the columns of A_k has no impact in R_k, and that

$$\Pi_k A_k A_k^T \Pi_k^T = R_k^T \underbrace{Q_k^T Q_k}_{I} R_k = R_k^T R_k.$$

Adding and deleting rows to A_k^T is allowed, which is respectively known as row updating and downdating the factorization. Therefore, what we have done is to adapt Saunders' techniques for square matrices [20] to matrices with more rows than columns, using the static data structure of George and Heath [4] but allowing a LINPACK-like row downdating on it. A MATLAB toolbox based on this technique has been developed and tested by Guerrero-García and Santos-Palomo [6] in a dual non-simplex active-set framework, and the interested reader can find the details of these row updating and downdating $\mathcal{O}(n^2)$ processes in [19].

This sparse orthogonal factorization allows us to obtain a basis of each fundamental subspace involved, because columns of R_k^T form a basis of $\mathcal{R}(\Pi_k A_k)$ and those of Z_k a basis of $\mathcal{N}(A_k^T)$, with

$$Z_k = \Pi_k^T \begin{bmatrix} Z^T \\ I \end{bmatrix}, \qquad \text{where} \qquad Z \doteq -R_\text{d}^T R_\text{i}^{-T}.$$

Thus, the independence check can be done as Björck and Duff [1] did, as follows:

$$v \in \mathcal{R}(A_k) \quad \Leftrightarrow \quad \forall s \in \mathcal{N}(A_k^T)\,,\ v^T s = 0 \quad \Leftrightarrow \quad Z_k^T v = \mathrm{O}$$

$$v \notin \mathcal{R}(A_k) \quad \Leftrightarrow \quad Z_k^T v \neq \mathrm{O} \qquad (\|Z_k^T v\| > \epsilon)$$

$$\Pi_k v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix},\ v_1 \in \mathbb{R}^{m_k} \quad \Rightarrow \quad Z_k^T v = Z v_1 + v_2 = -R_{\mathrm{d}}^T R_{\mathrm{i}}^{-T} v_1 + v_2$$

Furthermore, a basic solution of $A_k^T x = b_{\mathcal{B}}$ can be obtained by using some special "seminormal equations":

$$A_k A_k^T x = \Pi_k^T R_k^T R_k \Pi_k x = A_k b_{\mathcal{B}} \qquad \Rightarrow \qquad R_k^T(R_k(\Pi_k x)) = \Pi_k(A_k b_{\mathcal{B}}),$$

and analogously for $A_k y_{\mathcal{B}} = c$ with $y_{\mathcal{B}} \doteq A_k^T z$ but with $A_k A_k^T z = c$

$$A_k A_k^T z = \Pi_k^T R_k^T R_k \Pi_k z = c \qquad \Rightarrow \qquad R_k^T(R_k(\Pi_k z)) = \Pi_k c.$$

Note that $A_k A_k^T \geq 0$ in this special seminormal equations, whereas ordinary seminormal equations use $A_k^T A_k > 0$; moreover, fixed precision iterative refinement will improve the results. We do not have to write an "ad-hoc" routine for trapezoidal solves because zero diagonals of R are temporarily set to 1 before an usual triangular solve.

The same data structure is used for the crossover, in spite of the fact that A_k has not full-column rank in this case. Compatibility of $A_k^T x = b_{\mathcal{B}}$ is checked at each crossover iteration to decide whether a deletion or an exchange is performed.

4 The Modification Used in Phase-I

The main goal was to use the same sparse static data structure in all phases, so let us describe how we have to proceed during Phase-I. As previously noted, we have to work in Phase-I with the Cholesky factor R_k of $\sigma I_n + A_k A_k^T > 0$ (constant $\sigma > 0$), instead of with the Cholesky factor R_k of $A_k A_k^T \geq 0$.

Our proposal relies on the fact that the Cholesky factor of $\sigma I_n + A_k A_k^T$ coincides (barring signs) with the triangular factor of the QR factorization of $[A_k,\ \sqrt{\sigma} I_n]^T$, for

$$R_k^T R_k = \left[\, R_k^T, O_k \,\right] Q_k^T Q_k \begin{bmatrix} R_k \\ O_k \end{bmatrix} = \left[\, A_k, \sqrt{\sigma} I_n \,\right] \begin{bmatrix} A_k^T \\ \sqrt{\sigma} I_n \end{bmatrix} = \sigma I_n + A_k A_k^T.$$

The presence of the Marquardt-like term σI_n in the expression $\sigma I_n + A_k A_k^T$ to be factorized allows us to dispense with rank requisites on A_k, and this diagonal elements do not destroy the predicted static structure for $A_k A_k^T$ described below. Taking advantage of this sparse Marquardt-like regularization has also been done by Guerrero-García and Santos-Palomo [7] in a quite different context.

The crucial point is that in Phase-I we also work on top of the static structure of the Cholesky factor R of AA^T (because this structure coincides with that of the Cholesky factor of $\sigma I_n + AA^T$, cf. [10, p. 105]). An initialization from $R = \sqrt{\sigma} I_n$ in the diagonal of the static structure is done at the start of Phase-I, thus each row of A_k^T will be linearly dependent of the rows already present in the structure, and hence a sequence of Givens rotations has to be performed to annihilate all nonzero elements (and eventual fill-ins) of each row of A_k^T. Since all rows of $\sqrt{\sigma} I_n$ remain in the structure during Phase-I, downdating is done as described by Santos-Palomo and Guerrero-García [19].

Being able to use one initial ANALYSE phase is important when dealing with sparse matrix aspects. Previous work does not take advantage of this fact [16, p. 99], and then a dense implementation [14] based on the work of Fletcher and Powell [3] have been shown to be useful to obtain encouraging results in several continuation algorithms. A sparse updatable implementation of this Phase-I could also be done with the multifrontal orthogonal dynamic approach with hyperbolic downdating recently proposed by Edlund [2], although this would entail the use of regularization techniques in the remaining two phases of the hybrid algorithm.

5 Additional Features

A FORTRAN implementation of the updating process (without the downdating) can be found in the SPARSPAK package, which is currently being translated into C_{++} as described by George and Liu in [5]. The availability of this SPARSPAK$_{++}$ package would imply a rapid prototyping of a low-level implementation of the sparse proposal given here, and then meaningful comparative computational results for solving LPs could be obtained. As pointed out by a referee, this new LP approach is condemned to face fierce competition with professional implementations of both simplex and interior-point methods, but being able to update/downdate a factorization on a static data structure is crucial to become competitive in a distributed memory HPC environment.

A decrease of the computational effort when a refactorization (i.e., recomputing factorization from scratch) has to be done (because of the accumulation of rounding errors in its maintenance) can be accomplished by taking into account a suitable row ordering of A^T; note that doing so every iteration would prohibitively imply $\mathcal{O}(n^3)$ per iteration. The key point is that the column order of A_k does not affect the density of the Cholesky factor of $\sigma I_n + A_k A_k^T$ nor that of $A_k A_k^T$. Our MATLAB implementation takes advantage of this fact by using the sparse option of the `qr` command, which is an executable file (originally written in C, no source code available) implementing the updating process (without the downdating) referred to in §3. Moreover, we could also take advantage of the parallelizability of this refactorization.

Acknowledgements

The authors gratefully acknowledge the referees for their helpful corrections and remarks, which have improved the quality of the paper.

References

1. Åke Björck and Iain S. Duff. A direct method for the solution of sparse linear least squares problems. *Linear Algebra Appl.*, 34:43–67, 1980.
2. Ove Edlund. A software package for sparse orthogonal factorization and updating. *ACM Trans. Math. Software*, 28(4):448–482, December 2002.

3. Roger Fletcher and Michael J. D. Powell. On the modification of LDL^T factorizations. *Math. Comp.*, 29:1067–1087, 1974.
4. J. Alan George and Michael T. Heath. Solution of sparse linear least squares problems using Givens rotations. *Linear Algebra Appl.*, 34:69–83, 1980.
5. J. Alan George and Joseph W.-H. Liu. An object-oriented approach to the design of a user interface for a sparse matrix package. *SIAM J. Matrix Anal. Applics.*, 20(4):953–969, 1999.
6. Pablo Guerrero-García and Ángel Santos-Palomo. Solving a sequence of sparse compatible systems. Technical report MA-02-03, Department of Applied Mathematics, University of Málaga, November 2002. Submitted for publication.
7. Pablo Guerrero-García and Ángel Santos-Palomo. A sparse orthogonal factorization technique for certain stiff systems of linear ODEs. In J. Marín and V. Koncar (eds.), *Industrial Simulation Conference 2004 Proceedings*, pp. 17–19, June 2004.
8. Kaj Madsen, Hans B. Nielsen and Mustafa Ç. Pinar. A new finite continuation algorithm for linear programming. *SIAM J. Optim.*, 6(3):600–616, August 1996.
9. István Maros. *Computational Techniques of the Simplex Method*. Kluwer Academic Publisher, 2003.
10. Pontus Matstoms. Sparse linear least squares problems in optimization. *Comput. Optim. Applics.*, 7(1):89–110, 1997.
11. Nimrod Megiddo. On finding primal- and dual-optimal bases. *ORSA J. Computing*, 3(1):63–65, Winter 1991.
12. Katta G. Murty. *Linear Programming*. John Wiley and Sons, 1983.
13. Katta G. Murty. *Linear Complementarity, Linear and Nonlinear Programming*. Heldermann, 1988.
14. Hans B. Nielsen. AAFAC: A package of FORTRAN 77 subprograms for solving $A^T Ax = c$. Tech. report, Institute for Numerical Analysis, Technical University of Denmark, Lyngby, Denmark, March 1990.
15. Hans B. Nielsen. Computing a minimizer of a piecewise quadratic. Tech. report, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, September 1998.
16. Hans B. Nielsen. Algorithms for linear optimization, 2nd edition. Tech. report, Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, February 1999.
17. Ping-Qi Pan. A basis-deficiency-allowing variation of the simplex method for linear programming. *Computers Math. Applic.*, 36(3):33–53, August 1998.
18. Ángel Santos-Palomo and Pablo Guerrero-García. A non-simplex active-set method for linear programs in standard form. *Stud. Inform. Control*, 14(2), June 2005.
19. Ángel Santos-Palomo and Pablo Guerrero-García. Updating and downdating an upper trapezoidal sparse orthogonal factorization. Accepted for publication in *IMA J. Numer. Anal.*
20. Michael A. Saunders. Large-scale linear programming using the Cholesky factorization. Tech. report CS-TR-72-252, Department of Computer Science, Stanford University, Stanford, CA, January 1972.

Applying Parallel Direct Solver Techniques to Build Robust High Performance Preconditioners

Pascal Hénon[1], François Pellegrini[1], Pierre Ramet[1],
Jean Roman[1], and Yousef Saad[2]

[1] ScAlApplix Project, INRIA Futurs and LaBRI UMR 5800
Université Bordeaux 1, 33405 Talence Cedex, France
{henon,pelegrin,ramet,roman}@labri.fr
[2] Dept of computer Science & Eng.
University of Minnesota, 200 Union st. SE.
Minneapolis, MN 55155, USA
saad@cs.umn.edu

Abstract. The purpose of our work is to provide a method which exploits the parallel blockwise algorithmic approach used in the framework of high performance sparse direct solvers in order to develop robust preconditioners based on a parallel incomplete factorization. The idea is then to define an adaptive blockwise incomplete factorization that is much more accurate (and numerically more robust) than the scalar incomplete factorizations commonly used to precondition iterative solvers.

1 Introduction

Solving large sparse linear systems by iterative methods [18] has often been unsatisfactory when dealing with pratical “industrial" problems. The main difficulty encountered by such methods is their lack of robustness and, generally, the unpredictability and unconsistency of their performance when they are used over a wide range of different problems; some methods work quite well for certain types of problems but can fail completely on others.

Over the past few years, direct methods have made significant progress thanks to research studies on both the combinatorial analysis of Gaussian elimination process and on the design of parallel block solvers optimized for high-performance computers. It is now possible to solve real-life three-dimensional problems having in the order of several millions equations, in a very effective way with direct solvers. These is achievable by exploiting superscalar effects of modern processors and taking advantage of computer architectures based on networks of SMP nodes (IBM SP, DEC-Compaq, for example) [1,7,9,10,12]. However, direct methods will still fail to solve very large three-dimensional problems, due to the large amount of memory needed for these cases.

Some improvments to the classical scalar incomplete factorization have been studied to reduce the gap between the two classes of methods. In the context of domain decomposition, some algorithms that can be parallelized in an efficient way have been investigated in [14]. In [16], the authors proposed to couple incomplete factorization with a selective inversion to replace the triangular solutions (that are not as scalable as the factorization) by scalable matrix-vector multiplications. The multifrontal method has also been adapted for incomplete factorization with a threshold dropping in [11] or with a fill level dropping that measures the importance of an entry in terme of its updates [2]. In [3], the authors proposed a block ILU factorization technique for block tridiagonal matrices.

Our goal is to provide a method which exploits the parallel blockwise algorithmic approach used in the framework of high performance sparse direct solvers in order to develop robust parallel incomplete factorization based preconditioners [18] for iterative solvers.

The originality of our work is to use a supernodal algorithm what allows us to drop some blocks during the elimination process on the quotient graph. Unlike multifrontal approaches for which parallel threshold-based ILU factorization has been studied, the supernodal method permits to build at low cost a dense block structure for an ILU(k) factorization with an important fill-in. Indeed, using dense block formulation is crutial to achieved high performance computations.

The idea is then to define an adaptive blockwise incomplete factorization that is much more accurate (and numerically more robust) than the scalar incomplete factorizations commonly used to precondition iterative solvers. Such incomplete factorizations can take advantage of the latest breakthroughts in sparse direct methods and can therefore be very competitive in terms of CPU time due to the effective usage of CPU power. At the same time this approach does not suffer from the memory limitation encountered by direct methods. Therefore, we can expect to be able to solve systems in the order of hundred millions of unknowns on current platforms. Another goal of this paper is to analyze the chosen parameters that can be used to define the block sparse pattern in our incomplete factorization.

The remainder of the paper is organized as follows: the section 2 describes the main features of our method. We provide some experiments in section 3. At last we give some conclusions in section 4.

2 Methodology

The driving rationale for this study is that it is easier to incorporate incomplete factorization methods into direct solution software than it is to develop new incomplete factorizations. As a starting point, we can take advantage of the algorithms and the software components of PaStiX [10] which is a parallel high performance supernodal direct solver for sparse symmetric positive definite systems and for sparse unsymmetric systems with a symmetric pattern. These different components are (see Fig. 1):

1. the ordering phase, which computes a symmetric permutation of the initial matrix such that factorization process will exhibit as much concurrency as possible while incurring low fill-in. In this work, we use a tight coupling of the Nested Dissection and Approximate Minimum Degree algorithms [15];
2. the block symbolic factorization phase, which determines the block data structure of the factorized matrix associated with the partition resulting from the ordering phase. This structure consists of several column-blocks, each of them containing a dense diagonal block and a set of dense rectangular off-diagonal blocks [5];

3. the block repartitioning and scheduling phase, which refines the previous partition by splitting large supernodes in order to exploit concurrency within dense block computations in addition to the parallelism provided by the block elimination tree, both induced by the block computations in the supernodal solver. In this phase, we compute a mapping of the matrix blocks (that can be made by column block (1D) or block (2D)) and a static optimized scheduling of the computational and communication tasks according to BLAS and communication time models calibrated on the target machine. This static scheduling will drive the parallel factorization and the backward and forward substitutions [8,10].

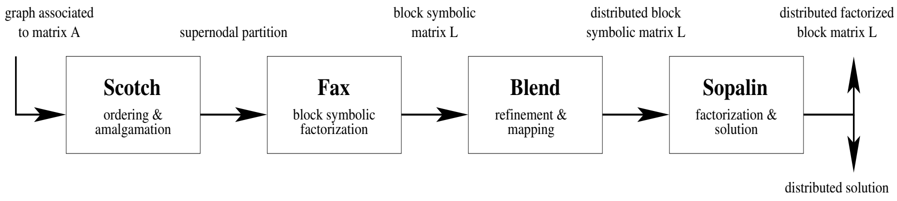


Fig. 1. Phases for the parallel complete block factorization

Our approach consists of *computing symbolically the block structure of the factors that would have been obtained with a complete factorization, and then deciding to drop off some blocks of this structure according to relevant criteria*. This incomplete factorization induced by the new sparse pattern is then used in a preconditioned GMRES or Conjugate Gradient solver [18]. Our main goal at this point is to achieve a significant reduction of the memory needed to store the incomplete factors while keeping enough fill-in to make the use of BLAS3 primitives cost-effective. Naturally, we must still have an ordering phase and a mapping and scheduling phase (this phase is modified to suit the incomplete factorization block computation) to ensure an efficient parallel implementation of the block preconditioner computation and of the forward and backward substitutions in the iterations. Then, we have the new processing chain given at Fig. 2.

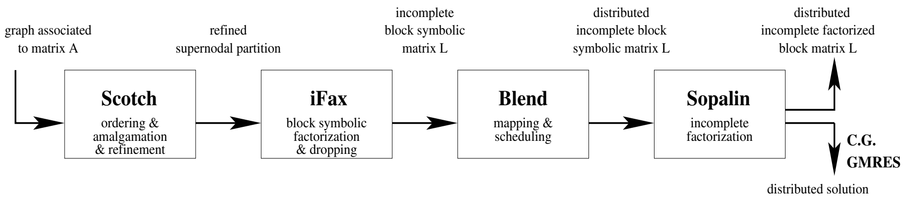

graph associated to matrix A
refined supernodal partition
incomplete block symbolic matrix L
distributed incomplete block symbolic matrix L
distributed incomplete factorized block matrix L
Scotch
ordering & amalgamation & refinement
iFax
block symbolic factorization & dropping
Blend
mapping & scheduling
Sopalin
incomplete factorization
C.G.
GMRES
distributed solution

Fig. 2. Phases for the parallel incomplete block factorization

The first crucial point is then to find a good initial partition that can be used in the dropping step after the block symbolic factorization. It cannot be the initial supernodal partition computed by the ordering phase (phase 1) because it would be too costly to consider the diagonal blocks as dense blocks like in a complete factorization. Therefore, we resort to a refined partition of this supernodal partition which will then define the elementary column blocks of the factors. We obtain the refined partition by splitting the column blocks issued from the supernodal partition according to a maximum blocksize parameter. This allows more options for dropping some blocks in the preconditioner. This blocksize parameter plays a key role in finding a good trade-off as described above. An important result from theoretical analysis is that if we consider nested dissection ordering based on separator theorems [13], and if we introduce some asymptotically refined partitions, one can show that the total number of blocks computed by the block symbolic factorization and the time to compute these blocks are quasi-linear (these quantities are linear for the supernodal partition).

The second crucial point concerns the various criteria that are used to drop some blocks from the blockwise symbolic structure induced by the refined partition. The dropping criterion we use is based on a generalization of the level-of-fill [18] metric that has been adapted to the elimination process on the quotient graph induced by the refined partition.

One of the most common ways to define a preconditioning matrix M is through Incomplete LU (ILU) factorizations. ILU factorizations are obtained from an approximate Gaussian elimination. When Gaussian elimination is applied to a sparse matrix A, a large number of nonzero elements may appear in locations originally occupied by zero elements. These fill-ins are often small elements and may be dropped to obtain approximate LU factorizations.

The simplest of these procedures, ILU(0) is obtained by performing the standard LU factorization of A and dropping all fill-in elements that are generated during the process. In other words, the L and U factors have the same pattern as the lower and upper triangular parts of A (respectively). More accurate factorizations denoted by ILU(k) and IC (k) have been defined which drop fill-ins according to their "levels". Level-1 fill-ins for example are generated from level-zero fill-ins (at most). So, for example, ILU(1) consists of keeping all fill-ins that have level zero or one and dropping any fill-in whose level is higher. We now provide a few details.

In level-based ILUs, originally introduced by Watts III [19], a *level-of-fill* is associated with every entry in the working factors L and U. Conceptually these factors together are the L and U parts of a certain working matrix A which initially contains the original matrix. At the start of the procedure, every zero entry has a level-of-fill of infinity and every nonzero entry has a level-of-fill of zero. Whenever an entry is modified by the standard Gaussian Elimination update

$$a_{ij} := a_{ij} - a_{ik} * a_{kj}/a_{kk}$$

its level-of-fill is updated by the formula

$$lev_{ij} = \min\{lev_{ij}, lev_{ik} + lev_{kj} + 1\}.$$

In practice, these levels are computed in a symbolic phase first and used to define the patterns of L and U a priori. As the level-of-fill increases, the factorization becomes more expensive but more accurate. In general, the robustness of the iterative solver will typically improve as the level-of-fill increases. It is common practice in standard preconditioned Krylov methods to use a very low level-of-fill, typically no greater than 2. Now it is easy to see what will be the main limitation of this approach: the level-of-fill is based entirely on the adjacency graph of the original matrix, and its definition implicitly assumes some form of diagonal dominance, in order for the dropped entries to be indeed small. It cannot work for matrices that are highly indefinite for example. There are two typical remedies, each with its own limitations. The first is to replace the level-of-fill strategy by one that is based on numerical values. This yields the class of ILUT algorithms [17,18]. Another is to resort to a block algorithm, i.e., one that is based on a block form of Gaussian elimination. Block ILU methods have worked quite well for harder problems, see for example [4,6].

In the standard block-ILU methods, the blocking of the matrix simply represents a grouping of sets of unknowns into one single unit. The simplest situation is when A has a natural block structure inherited from the blocking of unknowns by degrees of freedom at each mesh-point. Extending block ILU to these standard cases is fairly easy. It suffices to consider the quotient graph obtained from the blocking.

The situation with which we must deal is more complex because the block partitioning of rows varies with the block columns. An illustration is shown in Figure 3. The main difference between the scalar and the block formulation of the level-of-fill algorithm is that, in general, a block contribution may update only a part of the block (see figure 3(a)). As a consequence, a block is split according to its level-of-fill modification. Nevertheless, we only allow the splitting along the row dimension in order to preserve an acceptable blocksize (see figure 3(b)).

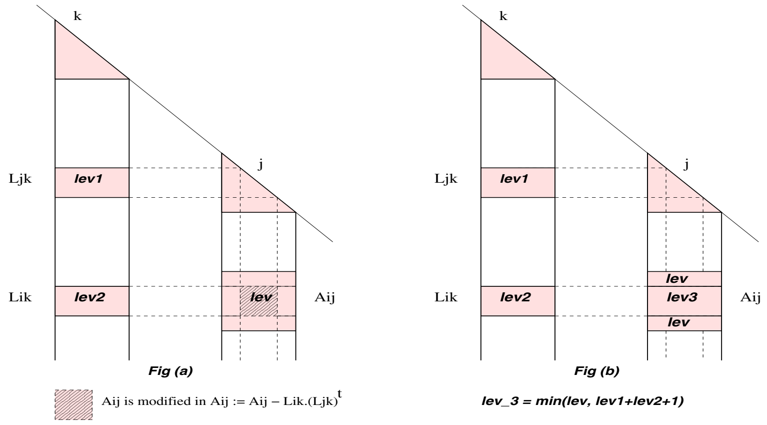

k
j
Ljk
lev1
Lik
lev2
lev
Aij
Fig (a)
▨ Aij is modified in Aij := Aij – Lik.(Ljk)^t
k
j
Ljk
lev1
Lik
lev2
lev
lev3
lev
Aij
Fig (b)
lev_3 = min(lev, lev1+lev2+1)

Fig. 3. Computation of the block level of fill-in the block elimination process

As shown in table 1, two consecutive levels-of-fill can produce a large increase of the fill ratio in the factors (NNZ_A is the number of non-zeros in A, NNZ_L is the number of non-zeros that would have beeen obtained with a direct factorization, and OPC_L is the number of operations required for the direct factorization).

Table 1. Fill rate for a 47x47x47 3D mesh (finite element, 27 connectivity)

level of fill	$\% NNZ_L$	$\times NNZ_A$	$\% OPC_L$
≤ 0	9.28	3.53	0.29
≤ 1	23.7	9.02	2.33
≤ 2	38.4	14.6	7.61
≤ 3	54.9	20.9	19.0
≤ 4	66.1	25.2	31.5
≤ 5	75.4	28.7	45.1
≤ 6	83.2	31.6	58.5
...	...	...	...
≤ 15	100	38.1	100

In order to choose intermediate ratios of fill between two levels, we have introduced a second criterion to drop some blocks inside a level-of-fill. We allow the possibility to choose a fill ratio according to the formula:

$$NNZ_{\text{prec}} = (1 - \alpha).NNZ_A + \alpha.NNZ_L$$

where $\alpha \in [0, 1]$ and NNZ_{prec} is the number of non-zeros in the factors for the incomplete factorization. To reach the exact number of non-zeros corresponding to a selected α, we consider the blocks allowed in the fill-in pattern in two steps. If we denote by NNZ_k the number of non-zeros obtained by keeping all the blocks with levels-of-fill $\leq k$, then we find the first value λ such that $NNZ_\lambda \geq NNZ_{\text{prec}}$. In a second step, until we reach NNZ_{prec}, we drop the blocks, among those having a level-of-fill λ, which undergo the fewest updates by previous eliminations.

3 Tests

In this section, we give an analysis of some first convergence and scalability results for practical large systems. We consider two difficult problems for direct solvers (see table 2). The AUDI matrix (symmetric) corresponds to an automotive crankshaft model and the MHD1 is a magnetohydrodynamic problem (unsymmetric). The ratio, between the number of non-zeros in the complete factor and the number of non-zeros in the initial matrix A is about 31 for the AUDI test case and about 67 for the MHD1 one.

Table 2. Description of our test problems

Name	Columns	NNZ_A	NNZ_L	OPC_L
AUDI	943695	39297771	1.21e+09	5.3e+12
MHD1	485597	24233141	1.62e+09	1.6e+13

Numerical experiments were run on a 28 NH2 IBM nodes (16 Power3+, 375Mhz, 1.5 Gflops, 16GB) located at CINES (Montpellier, France) with a network based on a Colony switch. All computations are performed in double precision and all time results are given in seconds. We use the PaStiX software with recent improvements such as an efficient MPI/Thread implementation to compute the preconditioner. The stopping criterion for GMRES iterations uses the relative residual norm and is set to 10^{-7}.

Table 3 presents results for different values of the fill rate parameter α for the AUDI problem. The blocksize (for the refined partition) is set to 8 and the results are performed on 16 processors.

Table 3. AUDI problem with blocksize=8

$\alpha = 0.1$			$\alpha = 0.2$			$\alpha = 0.3$		
$inc.fact.$	$nb.iter.$	$time/iter.$	$inc.fact.$	$nb.iter.$	$time/iter.$	$inc.fact.$	$nb.iter.$	$time/iter.$
24	429	0.71	39	293	1.30	55	279	1.64
$\times NNZ_A$	$\% \, OPC_L$	$tot.time$	$\times NNZ_A$	$\% \, OPC_L$	$tot.time$	$\times NNZ_A$	$\% \, OPC_L$	$tot.time$
3.77	0.51	**328.6**	6.75	2.91	**419.9**	9.75	5.97	**512.5**

$\alpha = 0.4$			$\alpha = 0.5$			$\alpha = 0.6$		
$inc.fact.$	$nb.iter.$	$time/iter.$	$inc.fact.$	$nb.iter.$	$time/iter.$	$inc.fact.$	$nb.iter.$	$time/iter.$
80	144	2.12	135	49	3.13	195	36	3.70
$\times NNZ_A$	$\% \, OPC_L$	$tot.time$	$\times NNZ_A$	$\% \, OPC_L$	$tot.time$	$\times NNZ_A$	$\% \, OPC_L$	$tot.time$
12.84	8.32	**385.3**	15.92	15.63	**288.4**	18.99	24.10	**328.2**

For each run we give:

– in the first line, the time to compute the incomplete factorisation ($inc.fact.$), the number of iterations ($nb.iter.$) and the time to perform one iteration ($time/iter.$);
– in the second line, the ratio between the number of non-zeros in the incomplete factors and the number of non-zeros in the matrix A ($\times NNZ_A$), the percentage of the number of operations to compute the incomplete factorization compared with the number of operations required for the direct factorization ($\% \, OPC_L$), and the total time ($tot.time$).

The same results for a blocksize set to 16 can be found in table 4. Theses results can be compared with the time required by the direct solver: on 16 processors, PaStiX needs about 482s to solve the problem and the solution has an accuracy (relative residual norm) about 10^{-15}.

Table 4. AUDI problem with blocksize=16

$\alpha = 0.1$			$\alpha = 0.2$			$\alpha = 0.3$		
$inc.fact.$	$nb.iter.$	$time/iter.$	$inc.fact.$	$nb.iter.$	$time/iter.$	$inc.fact.$	$nb.iter.$	$time/iter.$
11	214	0.84	19	196	1.20	40	177	1.87
$\times NNZ_A$	$\% \, OPC_L$	$tot.time$	$\times NNZ_A$	$\% \, OPC_L$	$tot.time$	$\times NNZ_A$	$\% \, OPC_L$	$tot.time$
5.29	0.97	**190.8**	6.50	2.16	**254.2**	9.57	6.82	**371.0**

$\alpha = 0.4$			$\alpha = 0.5$			$\alpha = 0.6$		
$inc.fact.$	$nb.iter.$	$time/iter.$	$inc.fact.$	$nb.iter.$	$time/iter.$	$inc.fact.$	$nb.iter.$	$time/iter.$
62	186	2.13	77	140	2.41	130	42	3.45
$\times NNZ_A$	$\% \, OPC_L$	$tot.time$	$\times NNZ_A$	$\% \, OPC_L$	$tot.time$	$\times NNZ_A$	$\% \, OPC_L$	$tot.time$
12.76	11.93	**458.2**	15.55	15.57	**414.4**	19.01	24.39	**274.9**

As expected, the time for the incomplete factorization and for the iterations increases with the fill rate parameter whereas the number of iterations decreases. We can see that the best result is obtained with α set to 0.1 and a blocksize set to 16. Thus, we can solve the problem 2.5 faster than with our direct solver and with only 17.2% of NNZ_L (about $5.3 \times NNZ_A$). We have also report results with higher values for the blocksize (32): block computations are more efficient but these blocksizes do not allow to drop enough entries to be competitive.

For next results the blocksize is set to 16 and α to 0.1. With such a fill rate parameter, the number of iterations for the MHD1 problem is small (5 iterations to reduce the residual norm by 10^{-7}). This problem is easier than the AUDI problem, and in that case, the advantage of our approach will be less important compared with traditional iterative solvers.

Table 5 shows that the run-time scalability is quite good for up to 64 processors for both the incomplete factorization and the iteration phase. We remind that the number of iterations is *independent* of the number of processors in our approach.

Table 5. Scalability results with $\alpha = 0.1$ and blocksize=16

Name	Number of processors						
	1	2	4	8	16	32	64
AUDI $inc.fact.$	90.9	52.5	29.2	15.7	10.6	5.9	3.3
AUDI $time/iter.$	10.5	5.56	3.06	1.49	0.84	0.45	0.33
MHD1 $inc.fact.$	48.1	26.2	15.6	9.1	5.1	3.0	2.2
MHD1 $time/iter.$	3.82	1.97	1.06	0.61	0.40	0.32	0.25

So on 64 processors, for a relative precision set to 10^{-7}, the total time is 74s what is twice faster than the 152s needed by the direct solver.

4 Conclusion

In conclusion, we have shown a methodology for bridging the gap between direct and iterative solvers by blending the best features of both types of techniques: low memory costs from iterative solvers and effective reordering and blocking from direct methods. The approach taken is aimed at producing robust parallel iterative solvers that can handle very large 3-D problems which arise from realistic applications. The preliminary numerical examples shown indicate that the algorithm performs quite well for such problems. Robustness relative to standard (non-block) preconditioners is achieved by extracting more accurate factorizations via higher amounts of fill-in. The effective use of hardware is enabled by a careful block-wise processing of the factorization, which yields fast factorization and preconditioning operations. We plan on performing a wide range of experiments on a variety of problems, in order to validate our approach and to understand its limitations. We also plan on studying better criteria for dropping certain blocks from the blockwise symbolic structure.

Acknowledgment

The work of the authors was supported from NSF grants INT-0003274. The work of the last author was also supported from NSF grants ACI-0305120 and by the Minnesota Supercomputing Institute.

References

1. P. R. Amestoy, I. S. Duff, S. Pralet, and C. Vömel. Adapting a parallel sparse direct solver to architectures with clusters of SMPs. *Parallel Computing*, 29(11-12):1645–1668, 2003.
2. Y. Campbell and T.A. Davis. Incomplete LU factorization: A multifrontal approach. *http://www.cise.ufl.edu/~davis/techreports.html*
3. T.F. Chang and P.S. Vassilevski. A framework for block ILU factorizations using block-size reduction. *Math. Comput.*, 64, 1995.
4. A. Chapman, Y. Saad, and L. Wigton. High-order ILU preconditioners for CFD problems. *Int. J. Numer. Meth. Fluids*, 33:767–788, 2000.
5. P. Charrier and J. Roman. Algorithmique et calculs de complexité pour un solveur de type dissections emboîtées. *Numerische Mathematik*, 55:463–476, 1989.
6. E. Chow and M. A. Heroux. An object-oriented framework for block preconditioning. Technical Report umsi-95-216, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1995.
7. Anshul Gupta. Recent progress in general sparse direct solvers. *Lecture Notes in Computer Science*, 2073:823–840, 2001.
8. P. Hénon. *Distribution des Données et Régulation Statique des Calculs et des Communications pour la Résolution de Grands Systèmes Linéaires Creux par Méthode Directe*. PhD thesis, LaBRI, Université Bordeaux I, France, November 2001.
9. P. Hénon, P. Ramet, and J. Roman. PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing*, 28(2):301–321, January 2002.
10. P. Hénon, P. Ramet, and J. Roman. Efficient algorithms for direct resolution of large sparse system on clusters of SMP nodes. In *SIAM Conference on Applied Linear Algebra, Williamsburg, Virginie, USA*, July 2003.
11. G. Karypis and V. Kumar. Parallel Threshold-based ILU Factorization. *Proceedings of the IEEE/ACM SC97 Conference*, 1997.
12. X. S. Li and J. W. Demmel. A scalable sparse direct solver using static pivoting. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, Texas, March 22-24, 1999.
13. R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM Journal of Numerical Analysis*, 16(2):346–358, April 1979.
14. M. Magolu monga Made and A. Van der Vorst. A generalized domain decomposition paradigm for parallel incomplete LU factorization preconditionings. *Future Generation Computer Systems*, Vol. 17(8):925–932, 2001.
15. F. Pellegrini, J. Roman, and P. Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency: Practice and Experience*, 12:69–84, 2000.
16. P. Raghavan, K. Teranishi, and E.G. Ng. A latency tolerant hybrid sparse solver using incomplete Cholesky factorization. *Numer. Linear Algebra*, 2003.
17. Y. Saad. ILUT: a dual threshold incomplete ILU factorization. *Numerical Linear Algebra with Applications*, 1:387–402, 1994.
18. Y. Saad. *Iterative Methods for Sparse Linear Systems, Second Edition*. SIAM, 2003.
19. J. W. Watts III. A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation. *Society of Petroleum Engineers Journal*, 21:345–353, 1981.

The Design of Trilinos

Michael A. Heroux and Marzio Sala

Sandia National Laboratories
{maherou,msala}@sandia.gov

Abstract. The Trilinos Project is an effort to facilitate the design, development, integration and ongoing support of mathematical software libraries within an object-oriented framework for the solution of large-scale, complex multi-physics engineering and scientific problems.
Trilinos is a two-level software structure, designed around a collection of *packages*. Each package focuses on a particular area of research, such as linear and nonlinear solver or algebraic preconditioners, and is usually developed by a small team of experts in this particular area of research. Packages exist underneath the Trilinos top level, which provides a common look-and-feel, including configuration, documentation, licensing, and bug-tracking.
In this paper we present the Trilinos design and an overview of the Trilinos packages. We discuss about the package interoperability and interdependence, and the Trilinos software engineering environment for developers. We also discuss how Trilinos facilitates high-quality software engineering practices that are increasingly required from simulation software.

1 Introduction

Trilinos [1,2], winner of an 2004 R&D 100 award, is a suite of parallel numerical solver libraries within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific applications in a production computing environment. The goal of the project is to facilitate the design, development, integration, and ongoing support of mathematical software libraries.

The emphasis of Trilinos is on developing robust and scalable algorithms in a software framework. Abstract interfaces are provided for flexible interoperability of packages while providing a full-featured set of concrete classes that implement all abstract interfaces. Furthermore, Trilinos includes an extensive set of tools and processes that support software engineering practices. Specifically, Trilinos provides the following:

– A suite of parallel object-oriented solver libraries for solving linear systems (using iterative methods and providing interfaces to third-party sequential and parallel direct solvers), nonlinear systems, defining incomplete factorizations, domain decomposition and multi-level preconditioners, solving eigensystem and time-dependent problems.
– Tools for rapid deployment of new packages.
– A scalable model for integrating new solver capabilities.
– An extensive set of software quality assurance (SQA) and software quality engineering (SQE) processes and tools for developers.

The fundamental concept in Trilinos is the *package*. Trilinos uses a two-level software structure designed around collections of packages. Each package is a numerical library (or collection of libraries) that is:

– Focused on important, state-of-the-art algorithms in its problem regime;
– Developed by a small team of domain experts;
– Self-contained, with minimal dependencies on other packages;
– Configurable, buildable, tested and documented on its own.

These packages can be distributed within Trilinos or separately. The Trilinos framework provides a common look-and-feel that includes configuration, documentation, licensing, and bug tracking. There are also guidelines for adding new packages to Trilinos.

Trilinos packages are primarily written in C++, but we provide some C and Fortran user interface support. Trilinos provides an open architecture that allows easy integration of other solver packages and we make our software available to the outside community via the GNU Lesser General Public License (LGPL) [3].

This paper is organized as follows. Section 2 gives an overview of the Trilinos packages. Section 3 describes the packages that can be used to define linear algebra components (like serial and distributed matrices and vectors), that can be used by all packages. Section 4 introduces to the NewPackage package, a jumpstart for new Trilinos packages. Section 5 outlines the difference between interoperability and interdependence between Trilinos packages. The Trilinos software engineering environment for solver developers is described in Section 6. Finally, Section 7 outlines the conclusions.

2 Overview of Trilinos Packages

The word “Trilinos” is Greek for “string of pearls”. It is a metaphor for the Trilinos architecture. Packages are strung together within Trilinos like pearls on a necklace. While each package is valuable and whole in its own right, the collection represents even more than the sum of its parts. Trilinos began as only three packages but has rapidly expanded. Table 1 lists the Trilinos packages with a brief description and their availability status. At the bottom of the table is the package count in each Trilinos release. General releases are available via automatic download. Limited releases are available upon request. A full description of Trilinos packages is available at the Trilinos Home Page [1].

A partial overview of what can be accomplished using Trilinos includes:

– Advanced serial dense or sparse matrices (Epetra);
– Advanced utilities for Epetra vectors and sparse matrices (EpetraExt);
– Templated distributed vectors and sparse matrices (Tpetra);
– Distributed sparse matrices (Epetra);
– Solve a linear system with preconditioned Krylov accelerators (AztecOO, Belos);
– Incomplete Factorizations (AztecOO, IFPACK);
– Multilevel preconditioners (ML);
– “Black-box” smoothed aggregation preconditioners (ML);

Table 1. Trilinos Package Descriptions and Availability

Package	Description	Release 3.1 (9/2003) 3.1 General	3.1 Limited	4 (5/2004) 4 General	4 Limited
Amesos	3rd Party Direct Solver Suite		X	X	X
Anasazi	Eigensolver package				X
AztecOO	Linear Iterative Methods	X	X	X	X
Belos	Block Linear Solvers				X
Epetra	Basic Linear Algebra	X	X	X	X
EpetraExt	Extensions to Epetra		X	X	X
Ifpack	Algebraic Preconditioners	X	X	X	X
Jpetra	Java Petra Implementation				X
Kokkos	Sparse Kernels			X	X
Komplex	Complex Linear Methods	X	X	X	X
LOCA	Bifurcation Analysis Tools	X	X	X	X
Meros	Segregated Preconditioners		X	X	X
ML	Multi-level Preconditioners	X	X	X	X
NewPackage	Working Package Prototype	X	X	X	X
NOX	Nonlinear solvers	X	X	X	X
Pliris	Dense direct Solvers			X	X
Teuchos	Common Utilities			X	X
TSFCore	Abstract Solver API			X	X
TSFExt	Extensions to TSFCore				X
Tpetra	Templated Petra				X
Totals		8	11	15	20

– One-level Schwarz preconditioner (overlapping domain decomposition) (AztecOO, IFPACK);
– Two-level Schwarz preconditioner, with coarse matrix based on aggregation (AztecOO and ML);
– Systems of nonlinear equations (NOX);
– Interface with various direct solvers, such as UMFPACK, MUMPS, SuperLU_DIST and ScaLAPACK (Amesos);
– Eigenvalue problems for sparse matrices (Anasazi);
– Complex linear systems (using equivalent real formulation) (Komplex);
– Segregated and block preconditioners (e.g., incompressible Navier-Stokes equations) (Meros);
– Light-weight interface to BLAS and LAPACK (Epetra);
– Templated interface to BLAS and LAPACK, arbitrary-precision arithmetic, parameters' list, smart pointers (Teuchos);

- Definition of abstract interfaces to vectors, linear operators, and solvers (TSF, TS-FCore, TSFExtended);
- Generation of test matrices (Triutils).

A detailed description of each package's functionalities is beyond the scope of this manuscript. In the next two sections, we focus our attention to the Petra object model, and to the NewPackage package.

3 The Petra Object Model

Scalable parallel implementation of algorithms is extremely challenging. Therefore, one of the most important features of Trilinos is its data model for global objects, called the Petra Object Model [6]. There are three implementation of this model, but the current production implementation is called Epetra [7]. Epetra is written for real-valued double-precision scalar field data, and restricts itself to a stable core of the C++ language standard. As such, Epetra is very portable and stable, and is accessible to Fortran and C users. Epetra combines in a single package:

- *Object-oriented, parallel C++ design*: Epetra has facilitated rapid development of numerous applications and solvers at Sandia because Epetra handles many of the complicated issues of working on a parallel, distributed-memory machine. Furthermore, Epetra provides a lightweight set of abstract interfaces and a shallow copy mode that allows existing solvers and data models to interact with Epetra without creating redundant copies of data.
- *High performance*: Despite the fact that Epetra provides a very flexible data model, performance has always been the top priority. Aztec [8] won an R&D 100 award in 1997 and has also been the critical performance component in two Gordon Bell finalist entries. Despite Epetra's much more general interfaces and data structures, for problems where Aztec and Epetra can be compared, Epetra is at least as fast and often faster than Aztec. Epetra also uses the BLAS and LAPACK wherever possible, resulting in excellent performance for dense operations.
- *Block algorithm support for next-generation solution capabilities*: The Petra Object Model and Epetra specifically provide support for multivectors, including distributed multivector operations. The level of support that Epetra provides for this feature is unique among available frameworks, and is essential for supporting next-generation applications that incorporate features such as sensitivity analysis and design optimization.
- *Generic parallel machine interfaces*: Epetra does not depend explicitly on MPI. Instead it has its own abstract interface for which MPI is one implementation. Experimental implementations for hybrid distributed/shared memory, PVM [9] and UPC [10] are also available or under development.
- *Parallel data redistribution*: Epetra provides a distributed object (DistObject) base class that manages global distributed objects. Not only does this base class provide the functionality for common distributed objects like matrices and vectors, it also supports distributed graphs, block matrices, and coloring objects. Furthermore, any class can derive from DistObject by implementing just a few simple methods to pack and unpack data.

– *Interoperability*: Application developers can use Epetra to construct and manipulate matrices and vectors, and then pass these objects to any Trilinos solver package. Furthermore, solver developers can develop many new algorithms relying on Epetra classes to handle the intricacies of parallel execution. Whether directly or via abstract interfaces, Epetra provides one of the fundamental interoperability mechanisms across Trilinos packages.

4 NewPackage Package: Fully Functional Package Prototype

One of the most useful features of Trilinos is the NewPackage package. This simple “Hello World" program is a full-fledged Trilinos package that illustrates:

– Portable build processes via Autoconf and Automake;
– Automatic documentation generation using Doxygen [18];
– Configuring for interaction with another package (Epetra) including the use of M [4,19] macros to customize package options;
– Regression testing using special scripts in the package that will be automatically executed on regression testing platforms.

In addition, Trilinos provides a NewPackage website that can be easily customized, requiring only specific package content. This website is designed to incorporate the reference documentation that is generated by Doxygen from the NewPackage source code.

Using NewPackage as a starting point, developers of mathematical libraries are able to become quickly and easily interoperable with Trilinos. It is worth noting that a package becomes Trilinos-interoperable without sacrificing its independence. This fact has made Trilinos attractive to a number of developers. As a result, Trilinos is growing not only by new development projects, but also by incorporation of important, mature projects that want to adopt modern software engineering capabilities. This is of critical importance to applications that depend on these mature packages.

5 Package Interoperability vs. Interdependence

Trilinos provides configuration, compilation and installation facilities via GNU Autoconf [4] and Automake [5]. These tools make Trilinos very portable and easy to install. The Trilinos-level build tools allow the user to specify which packages should be configured and compiled via `--enable-package_name` arguments to the configure command.

A fundamental difference between Trilinos and other mathematical software frameworks is its emphasis on interoperability. Via abstract interfaces and configure-time enabling, packages within Trilinos can interoperate with each other without being interdependent. For example, the nonlinear solver package NOX needs some linear algebra support, but it does not need to know the implementation details. Because of this, it specifies the interface it needs. At configure time, the user can specify one or more of three possible linear algebra implementations for NOX:

– `--enable-nox-lapack`: compile NOX lapack interface libraries
– `--enable-nox-epetra`: compile NOX epetra interface libraries
– `--enable-nox-petsc`: compile NOX PETSc interface libraries

Alternatively, because the NOX linear algebra interfaces are abstract, users can provide their own implementation and build NOX completely independent from Trilinos.

Another notable example is ML [20]. ML is the multilevel preconditioning package of Trilinos, and it can be used to define multigrid/multilevel preconditioners. ML contains basic Krylov accelerators and a variety of smoothers (like Jacobi, Gauss-Seidel, and polynomial smoothers), and as such it can be configured and used as a stand-alone package. That is, ML does not *depend* on any other Trilinos packages. However, ML can easily *interoperate* with other Trilinos packages, since light-weight conversions from the ML proprietary format to Epetra-format (and vice-versa) exist[1]. For instance, ML can:

– use the incomplete factorizations of AztecOO and IFPACK as smoothers;
– exploit the eigensolvers of Anasazi when an eigen-computation is required;
– take advantage of the Amesos interface to third-party libraries;
– use the general-purpose tools provided by Teuchos to define better interfaces.

Other packages can take advantages of ML as well: for example, an ML object can be used to define a preconditioner for an AztecOO linear system.

For ML applications, interoperability is extremely important. Multilevel preconditioners are based on a variety of components (possibly, a different aggregation scheme, pre-smoother and post-smoother for each level, and the coarse solver), and it is often of important to test different combinations of parameters, or add new components. To this aim, several Trilinos packages can be used to extend ML. Even if the kernel features of ML remain the same, independently of the available Trilinos packages, ML itself can be less effective as a stand-alone package, as it is as part of the Trilinos project.

6 Trilinos Software Engineering Environment for Solver Developers

Trilinos package developers enjoy a rich set of tools and well-developed processes that support the design, development and support of their software. The following resources are available for each package team:

– *Repository (CVS)*: Trilinos uses CVS [11] for source control. Each package exists both as an independent module as well as part of the larger whole.
– *Issue Tracking (Bugzilla)*: Trilinos uses Mozilla's Bugzilla [12] for issue tracking. This includes the ability to search all previous bug reports and to automatically track (via email notifications) submissions.

[1] If required, ML interoperates with packages outside the Trilinos project. For example, METIS and ParMETIS [21] can be used to define the aggregates in smoothed aggregation, or ParaSails [22] can be called to compute smoothers based on sparse approximate inverses.

- *Communication (Mailman)*: Trilinos uses Mailman [3] for managing mailing lists. There are Trilinos mailing lists as well as individual package mailing lists as follows. Each package has low volume lists for announcements as well as a general users mailing list. Further, each package also has developer-focused lists for developer discussion, tracking of CVS commits, and regression test results. Finally, there is a mailing list for the leaders of the packages, and these leaders also have monthly teleconferences where any necessary Trilinos-level decisions are made.

In addition to the above package-specific infrastructure, the following resources are also available to package developers:

- *Debugging (Bonsai)*: Mozilla Bonsai [4] is a web-based GUI for CVS, allowing for queries and easy comparison of different versions of the code. Bonsai is integrated with the Trilinos CVS and Bugzilla databases.
- *Jumpstart (NewPackage package)*: One of the fundamental contributions of Trilinos is “NewPackage". This is a legitimate package within Trilinos that does “Hello World". We discuss it in more detail below.
- *Developer Documentation*: Trilinos provides an extensive developers guide [15] that provides new developers with detailed information on Trilinos resources and processes. All Trilinos documents are available online at the main Trilinos home page1.
- *SQA/SQE support*: The Advanced Scalable Computing Initiative (ASCI) program within the Department of Energy (DOE) has strong software quality engineering and assurance requirements. At Sandia National Laboratories this has led to the development of 47 SQE practices that are expected of ASCI-funded projects [16]. Trilinos provides a special developers guide17 for ASCI-funded Trilinos packages. For these packages, 32 of the SQE practices are directly addressed by Trilinos. The remaining 15 are the responsibility of the package development team. However, even for these 15, Trilinos provides significant support.

To summarize, the two-level design of Trilinos allows package developers to focus on only those aspects of development that are unique to their package and still enjoy the benefits of a mature software engineering environment.

7 Summary

Computer modeling and simulation is a credible third approach to fundamental advances in science and engineering, along with theory and experimentation. A critical issue for continued growth is access to both new and mature mathematical software on high performance computers in a robust, production environment. Trilinos attempts to address this issue in the following ways: Provides a rapidly growing set of unique solver capabilities: Many new algorithms are being implemented using Trilinos. Trilinos provides an attractive development environment for algorithm specialists. Provides an independent, scalable linear algebra package: Epetra provides extensive tools for construction and manipulation of vectors, graphs and matrices, and provides the default implementation of abstract interfaces across all other Trilinos packages. Supports SQA/SQE requirements: First solver framework to provide explicitly documented processes for SQA/SQE.

An essential feature for production computing. Provides an innovative architecture for modular, scalable growth: The Trilinos package concept and the NewPackage package provide the framework for rapid development of new solvers, and the integration and support of critical mature solvers. Is designed for interoperability: Designed "from the ground up" to be compatible with existing code and data structures. Trilinos is the best vehicle for making solver packages, whether new or existing, interoperable with each other. Uses open source tools: Extensive use of third-party web tools for documentation, versioning, mailing lists, and bug tracking. To the best of our knowledge, no competing numerical libraries offer anything nearly as comprehensive.

Acknowledgments

The authors would like to acknowledge the support of the ASCI and LDRD programs that funded development of Trilinos.

References

1. The Trilinos Home Page: `http://software.sandia.gov/trilinos`, 16-Feb-04.
2. Michael A. Heroux, et. al., An Overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories, 2003.
3. The GNU Lesser General Public License: `http://www.gnu.org/copyleft/lesser.html`, 16-Feb-04.
4. GNU Autoconf Home Page: `http://www.gnu.org/software/autoconf`, 16-Feb-04.
5. GNU Automake Home Page: `http://www.gnu.org/software/automake`, 16-Feb-04.
6. Erik Boman, Karen Devine, Robert Heaphy, Bruce Hendrickson and Michael A. Heroux. LDRD Report: Parallel Repartitioning for Optimal Solver Performance. Technical Report SAND2004-XXXX, Sandia
7. Epetra Home Page: `http://software.sandia.gov/trilinos/packages/epetra`, 16-Feb-04.
8. Ray S. Tuminaro, Michael A. Heroux, Scott A. Hutchinson and John N. Shadid. Official Aztec User's Guide, Version 2.1. Technical Report SAND99-8801J, Sandia National Laboratories, 1999.
9. PVM Home Page: `http://www.csm.ornl.gov/pvm`, 16-Feb-04.
10. UPC Home Page: `http://upc.gwu.edu/`, 16-Feb-04.
11. GNU CVS Home Page: `http://www.gnu.org/software/cvs`, 16-Feb-04.
12. Bugzilla Home Page: `http://www.bugzilla.org`, 16-Feb-04.
13. GNU Mailman Home Page: `http://www.gnu.org/software/mailman`, 16-Feb-04.
14. The Bonsai Project Home Page:, `http://www.mozilla.org/projects/bonsai`, 16-Feb-04.
15. Michael A. Heroux, James M. Willenbring and Robert Heaphy, Trilinos Developers Guide, Technical Report SAND2003-1898, Sandia National Laboratories, 2003.
16. J. Zepper, K Aragon, M. Ellis, K. Byle and D. Eaton, Sandia National Laboratories ASCI Applications

17. Michael A. Heroux, James M. Willenbring and Robert Heaphy, Trilinos Developers Guide Part II: ASCI Software Quality Engineering Practices, Version 1.0. Technical Report SAND2003-1899, Sandia National Laboratories, 2003.
18. Doxygen Home Page: `http://www.stack.nl/ dimitri/doxygen`, 16-Feb-04.
19. GNU M4 Home Page: `http://www.gnu.org/software/m4`, 16-Feb-04.
20. M. Sala and J. Hu and R. Tuminaro, ML 3.0 Smoothed Aggregation User's Guide, Technical Report SAND2004-2195, Sandia National Laboratories, 2004.
21. G. Karypis and V. Kumar, ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, Technical Report 97-060, Department of Computer Science, University of Minnesota, 1997.
22. E. Chow, ParaSails User's Guide, Technical Report UCRML-MA-137863, Lawrence Livermore National Laboratory, 2000.

Software Architecture Issues in Scientific Component Development

Boyana Norris

Mathematics and Computer Science Division, Argonne National Laboratory
9700 South Cass Ave., Argonne, IL 60439, USA
norris@mcs.anl.gov

Abstract. Commercial component-based software engineering practices, such as the CORBA component model, Enterprise JavaBeans, and COM, are well-established in the business computing community. These practices present an approach for managing the increasing complexity of scientific software development, which has motivated the Common Component Architecture (CCA), a component specification targeted at high-performance scientific application development. The CCA is an approach to component development that is minimal in terms of the complexity of component interface requirements and imposes a minimal performance penalty. While this lightweight specification has enabled the development of a number of high-performance scientific components in several domains, the software design process for developing component-based scientific codes is not yet well defined. This fact, coupled with the fact that component-based approaches are still new to the scientific community, may lead to an ad hoc design process, potentially resulting in code that is harder to maintain, extend, and test and may negatively affect performance. We explore some concepts and approaches based on widely accepted software architecture design principles and discuss their potential application in the development of high-performance scientific component applications. We particularly emphasize those principles and approaches that contribute to making CCA-based applications easier to design, implement, and maintain, as well as enabling dynamic adaptivity with the goal of maximizing performance.

1 Introduction

Component-based software engineering (CBSE) practices are well established in the business computing community [1,2,3]. Component approaches often form the heart of architectural software specifications. Garlan and Shaw [4] define a software architecture as a specification of the overall system structure, such as "gross organization and global control structure; protocols for communication, synchronization, and data access; assignment of functionality to design elements, physical distribution; composition of design elements; scaling and performance; and selection among design alternatives." This decade-old definition largely holds today.

Recently, the emergence of a component specification targeted at high-performance computing has led to initial experimentation with CBSE in scientific software. The Common Component Architecture Forum [5,6] has produced a component architecture specification [7] that is semantically similar to other component models, with an emphasis on minimizing programming requirements and not imposing a significant performance penalty as a result of using components. The CCA specification is defined by using SIDL (Scientific Interface Definition Language), which provides language interoperability without sacrificing performance.

CCA employs the notion of *ports*, public interfaces that define points of interaction between components. There are two types of ports: *provides* and *uses*, used to specify functionality provided by a component, and to access functionality of other components that provide the matching port type. To be CCA-compliant, a component must implement the `gov.cca.Component` interface, which uses the method `setServices` to pass a reference to a framework services handle to the component. This handle subsequently is used for most framework interactions, such as registering *uses* and *provides* ports and obtaining and releasing port handles from the framework. The `setServices` method is normally invoked by the framework after the component has been instantiated. A CCA framework provides an implementation of the `Services` interface and performs component management, including dynamic library loading, component instantiation, and connection and disconnection of compatible ports.

In addition to the benefits of a component system for managing software complexity, the CCA offers new challenges and opportunities. In the remainder of this paper, we present some software architecture principles and approaches that build on the current CCA specification and help better define the component design and development process, as well as enable better dynamic adaptivity in scientific component applications. Going beyond the software architecture specification, we briefly discuss other important aspects of scientific component development, including code generation, implementation, compilation, and deployment.

2 Software Architecture Principles

In this section we examine software design approaches in the context of their applicability to high-performance scientific component development. One of the main distinctive features of the CCA is its minimal approach to component specification. This was partly motivated by the need to keep component overhead very low. Another motivation is to make the software design process more accessible to scientists who are concerned with the human overhead of adopting an approach new to the scientific computing community. The result is a working minimal-approach component model that has been used successfully in diverse applications [8,9,10,11]. The CCA specification incorporates a number of principles that Buschmann et al. [12] refer to as *architecture-enabling* principles, including abstraction, encapsulation, information hiding, modularization, coupling and cohesion, and separation of interface and implementation.

While the minimal CCA approach does make scientific component development possible, it is not a complete design methodology, as is, for example, the Booch method [13]. Furthermore, other than the minimal requirements of the component interface and framework-component interactions, the actual component design is not dictated by any set of rules or formal specifications. This makes the CCA very general and flexible but imposes a great burden on the software developer, who in many cases has no previous component- or object-oriented design experience. We believe that a number of widely accepted software architecture principles, such as those summarized in [12,14], can contribute greatly to all aspects of scientific application development—from interface specification to component quality-of-service support. These *enabling technologies* [12] can be used as guiding design principles for the scientific component developer. Moreover, they can enable the development of tools that automate many of the steps in the component development life cycle, leading to shorter development cycles and to applications that are both more robust and better performing.

Although the following software architecture principles are applicable to a wide range of domains, we focus on their impact on high-performance components, with an emphasis on adaptability. We briefly examine each approach and discuss an example context of its applicability in scientific component development.

Separation of Concerns. Currently the CCA provides a basic specification for components and does not deal directly with designing ports and components so that different or unrelated responsibilities are separate from each other and that different roles played by a component in different contexts are independent and separate from each other within the component. Several domain-specific interface definition efforts are under way, such as interfaces for structured and unstructured mesh access and manipulation, linear algebra solvers, optimization, data redistribution, and performance monitoring. These common interfaces ideally would be designed to ensure a clear separation of concerns. In less general situations, however, this principle is often unknown or ignored in favor of quicker and smaller implementation. Yet clearly separating different or unrelated responsibilities is essential for achieving a flexible and reliable software architecture. For example, performance monitoring functionality is often directly integrated in the implementation of components, making it difficult or impossible to change the amount, type, and frequency of performance data gathered. A better design is to provide ports or components that deal specifically with performance monitoring, such as those described in [15], enabling the use of different or multiple monitor implementations without modifying the implementation of the client components, as well as generating performance monitoring ports automatically. Similar approaches can be used for other types of component monitoring, such as that needed for debugging. Although in some cases the separation of concerns is self-evident, in others it is not straightforward to arrive at a good design while maintaining a granularity that does not deteriorate the performance of the application as a whole. For example, many scientific applications rely on a discretized representation of the problem domain, and it would seem like a good idea to extract the mesh management functionality into separate components. Depending on the needs of the application, however, accessing the mesh frequently through a fine-grained port interface may have prohibitive overhead. That is, not to say that mesh interfaces are doomed to bad performance; rather, care must be taken in the design of the interface, as well as the way in which it is used, in order to avoid loss of performance.

Separation of Policy and Implementation. A *policy* component deals with context-sensitive decisions, such as assembly of disjoint computations as a result of selecting parameter values. An *implementation* component deals only with the execution of a fully specified algorithm, without being responsible for making context-dependent decisions internally. This separation enables the implementation of multimethod solution methods, such as the composite and adaptive linear solvers described in [16,17]. Initial implementations of our multimethod linear system solution methods were not in component form (rather, they were based directly on PETSc [18]), and after implementing a few adaptive strategies, adding new ones became a very complex and error-prone task. The reason was that, in order to have nonlinear solver context information in our heuristic implementation of the adaptive linear solution, we used (or rather, "abused") the PETSc nonlinear user-defined monitor routine, which is invoked automatically via a call-back mechanism at each nonlinear iteration. Without modifying PETSc itself, this was the best way both to have context information about the nonlinear solution, while monitoring the performance and controlling the choice of linear solvers. While one can define the implementation in a structured way, one is still limited by the fact that the monitor is a single function, under which all adaptive heuristics must be implemented. Recently we have reworked our implementation to use well-defined interfaces for performance data management and adaptive linear solution heuristics. This separation of policy (the adaptive heuristic, which selects linear solvers based on the context in the nonlinear solver) and implementation (the actual linear solution method used, such as GMRES) not only has lead to a simpler and cleaner design but has made it possible to add new adaptive heuristics with a negligible effort compared to the noncomponent version. Design experiences, such as this nonlinear PDE solution example, can lead to guidlines or "best practices" that can assist scientists in achieving separation of policy and implementation in different application domains.

Reflection. The *Reflection* architectural pattern enables the structure and behavior of software systems to be changed dynamically [12]. Although Reflection is not directly supported by the CCA specification, similar capabilities can be provided in some cases. For example, for CCA components implemented with SIDL, interface information is available in SIDL or XML format. Frameworks or automatically generated ports (which provide access to component metainformation) can be used to provide Reflection capabilities in component software. Reflection also can be used to discover interfaces provided by components, a particularly useful capability in adaptive algorithms where a selection of a suitable implementation must be made among several similar components. For example, in an adaptive linear system solution, several solution methods implement a common interface and can thus be substituted at runtime, but knowing more implementation-specific interface details may enable each linear solver instance to be tuned better by using application-specific knowledge. We note, however, that while reflection can be useful in cases such as those cited, it may negatively affect performance if used at a very fine granularity. As with other enabling architectural patterns, one must be careful to ensure that it is applied only at a coarse-enough granularity to minimize the impact on performance.

While the omission of formal specifications for these and other features makes the CCA approach general and flexible, it also increases the burden on the scientific programmer. While in theory it is possible to design and implement CCA components in a framework-independent fashion, in practice the component developer is responsible for ensuring that a component is written and built in such a way that it can be used in a single framework that is usually chosen a priori[1].

[1] The CCA Forum has defined a number of interfaces for framework interoperability, so that while components are written in a way that may not be portable across frameworks, interaction between components executing in different frameworks is possible in some cases.

3 Implementation and Deployment

Implementation and deployment of CCA components are potentially complex and time-consuming tasks compared to the more traditional library development approaches. Therefore, CCA Forum participants have recently begun streamlining the component creation and build process. For example, CHASM [19] is being used to automate some of the steps required to create language-independent components from legacy codes or from scratch [20]. We have also recently begun automating the build process by generating most of the scripts needed to create a build system based on GNU tools, such as automake and autoconf [21,22]. Currently we are integrating the component generation tools based on CHASM [19] and Babel [23] with the build automation capabilities. Other efforts are making the code generation, build automation, and deployment support tools easy to use, extensible, portable, and flexible. Until recently, little attention was given to the human overhead of component design and implementation. Many of the above-mentioned efforts aim to increase the *software developers'* efficiency. Most ongoing work focuses on the tasks of generating ports and components from existing codes, generating code from existing interfaces, compiling, and deploying components. At this point, however, little is available on the techniques for *designing* ports and components. We believe that the architecture-enabling principles mentioned in this paper, as well as others, can help guide the design of scientific applications. Furthermore, we hope that in the near future, when more and more component scientific applications go through this design and implementation process, domain-specific patterns will emerge that will lead to the same type of improvement in the software development approaches and efficiency that design and architectural patterns have led to in the business software development community.

Scientific applications may need to execute in environments with different capabilities and requirements. For example, in some cases it is possible and desirable to build components as dynamic libraries that are loaded by the framework at runtime. In other cases, the computing environment, such as a large parallel machine without a shared file system, makes the use of dynamic libraries onerous, and one statically linked executable for the assembled component application is more appropriate. Being able to support both dynamic and static linking in an application makes the build process complex and error-prone, thus necessitating automation. Furthermore, support of multiple deployment modes (e.g., source-based, RPM or other binary formats) is essential for debugging, distribution, and support of multiple hardware platforms. Many of these issues can be addressed by emerging CCA-based tools that generate the required component meta-information, configuration, build, and execution scripts.

4 QoS-Enabled Software Architecture

As more functionally equivalent component implementations become available, the task of selecting components and assembling them into an application with good overall performance becomes complex and possibly intractable manually. Furthermore, as computational requirements change during the application's execution, the initial selection of components may no longer ensure good overall performance. Some of the considerations that influence the choice of implementation are particular to parallel codes, for example, the scalability of a given algorithm. Others deal with the robustness of the solution method, or its convergence speed (if known) for a certain class of problems.

Recent work [24,25,26] on computational quality of service (QoS) for scientific components has defined a preliminary infrastructure for the support of performance-driven application assembly and adaptation. The CCA enables this infrastructure to augment the core specifications, and new tools that exploit the CCA ports mechanism for performance data gathering and manipulations are being developed [15]. Related work in the design and implementation of QoS support in component architectures includes component contracts [27], QoS aggregation models [28], and QoS-enabled distributed component computing [29,30].

5 Conclusion

We have examined a number of current challenges in handling the entire life cycle of scientific component application development. We outlined some software architecture ideas that can facilitate the software design process of scientific applications. We discussed the need and ongoing work in automating different stages of component application development, including code generation from language-independent interfaces, automatic generation of components from legacy code, and automating the component build process. We noted how some of the software architecture ideas can be used to support adaptivity in applications, enabling computational quality-of-service support for scientific component applications. Many of these ideas are in the early stages of formulation and implementation and will continue to be refined and implemented as command-line or graphical tools, components, or framework services.

Acknowledgments

Research at Argonne National Laboratory was supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract W-31-109-ENG-38. We thank Matt Knepley, Lois McInnes, Paul Hovland, and Kate Keahey of Argonne National Laboratory for many of the ideas that led to this work, Sanjukta Bhowmick and Padma Raghavan of the Pennsylvania State University for their ongoing contributions, and the CCA Forum for enabling this collaborative effort.

References

1. Anonymous: CORBA component model. http://www.omg.org/technology/documents/formal/components.htm (2004)
2. Anonymous: Enterprise JavaBeans downloads and specifications. http://java.sun.com/products/ejb/docs.html (2004)
3. Box, D.: Essential COM. Addison-Wesley Pub. Co. (1997)
4. Garlan, D., Shaw, M.: An Introduction to Software Architecture. In: Advances in Software Engineering and Knowledge Engineering. World Scientific Publishing Company (1993)

5. Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L. C., Parker, S., Smolinski, B.: Toward a common component architecture for high-performance scientific computing. In: Proceedings of High Performance Distributed Computing. (1999) 115–124
6. Common Component Architecture Forum: CCA Forum website. `http://www.cca-forum.org` (2004)
7. CCA Forum: CCA specification. `http://cca-forum.org/specification/` (2003)
8. Norris, B., Balay, S., Benson, S., Freitag, L., Hovland, P., McInnes, L., Smith, B.: Parallel components for PDEs and optimization: Some issues and experiences. Parallel Computing **28 (12)** (2002) 1811–1831
9. Lefantzi, S., Ray, J.: A component-based scientific toolkit for reacting flows. In: Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics, Boston, Mass., Elsevier Science (2003)
10. Benson, S., Krishnan, M., McInnes, L., Nieplocha, J., Sarich, J.: Using the GA and TAO toolkits for solving large-scale optimization problems on parallel computers. Technical Report ANL/MCS-P1084-0903, Argonne National Laboratory (2003)
11. Larson, J. W., Norris, B., Ong, E. T., Bernholdt, D. E., Drake, J. B., Elwasif, W .R., Ham, M. W., Rasmussen, C. E., Kumfert, G., Katz, D. S., Zhou, S., DeLuca, C., Collins, N. S.: Components, the Common Component Architecture, and the climate/weather/ocean community. In: 84th American Meteorological Society Annual Meeting, Seattle, Washington, American Meteorological Society (2004)
12. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons Ltd. (1996)
13. Booch, G.: Unified method for object-oriented development Version 0.8. Rational Software Corporation (1995)
14. Medvidovic, N., Taylor, R. N.: A classification and comparison framework for software architecture description languages. IEEE Transactions on Software Engineering **26** (2000)
15. Shende, S., Malony, A. D., Rasmussen, C., Sottile, M.: A performance interface for component-based applications. In: Proceedings of International Workshop on Performance Modeling, Evaluation and Optimization, International Parallel and Distributed Processing Symposium (2003)
16. Bhowmick, S., Raghavan, P., McInnes, L., Norris, B.: Faster PDE-based simulations using robust composite linear solvers. Future Generation Computer Systems **20** (2004) 373–387
17. McInnes, L., Norris, B., Bhowmick, S., Raghavan, P.: Adaptive sparse linear solvers for implicit CFD using Newton-Krylov algorithms. In: Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics, Massachusetts Institute of Technology, Boston, USA, June 17-20, 2003
18. Balay, S., Buschelman, K., Gropp, W., Kaushik, D., Knepley, M., McInnes, L., Smith, B. F., Zhang, H.: PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory (2003). `http://www.mcs.anl.gov/petsc`.
19. Rasmussen, C. E., Lindlan, K. A., Mohr, B., Striegnitz, J.: Chasm: Static analysis and automatic code generation for improved Fortran 90 and C++ interoperability. In: 2001 LACSI Symposium (2001)
20. Rasmussen, C. E., Sottile, M. J., Shende, S. S., Malony, A. D.: Bridging the language gap in scientific computing: the chasm approach. Technical Report LA-UR-03-3057, Advanced Computing Laboratory, Los Alamos National Laboratory (2003)
21. Bhowmick, S.: Private communication. Los Alamos National Laboratory (2004)
22. Wilde, T.: Private communication. Oak Ridge National Laboratory (2004)
23. Anonymous: Babel homepage. http://www.llnl.gov/CASC/components/babel.html (2004)

24. Trebon, N., Ray, J., Shende, S., Armstrong, R.C., Malony, A.: An approximate method for optimizing HPC component applications in the presence of multiple component implementations. Technical Report SAND2003-8760C, Sandia National Laboratories (2004) Also submitted to 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments, held during the 18th International Parallel and Distributed Computing Symposium, 2004, Santa Fe, NM, USA.
25. Hovland, P., Keahey, K., McInnes, L. C., Norris, B., Diachin, L. F., Raghavan, P.: A quality of service approach for high-performance numerical components. In: Proceedings of Workshop on QoS in Component-Based Software Engineering, Software Technologies Conference, Toulouse, France (2003)
26. Norris, B., Ray, J., Armstrong, R., McInnes, L.C., Bernholdt, D.E., Elwasif, W.R., Malony, A.D., Shende, S.: Computational quality of service for scientific components (2004). In: Proceedings of International Symposium on Component-Based Software Engineering (CBSE7), Edinburgh, Scotland.
27. Beugnard, A., Jézéquel, J.M., Plouzeau, N., Watkins, D.: Making components contract aware. IEEE Computer (1999) 38–45
28. Gu, X., Nahrstedt, K.: A scalable QoS-aware service aggregation model for peer-to-peer computing grids. In: Proceedings of High Performance Distributed Computing. (2002)
29. Loyall, J. P., Schantz, R. E., Zinky, J. A., Bakken, D. E.: Specifying and measuring quality of service in distributed object systems. In: Proceedings of the First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC '98). (1998)
30. Raje, R., Bryant, B., Olson, A., Auguston, M., , Burt, C.: A quality-of-service-based framework for creating distributed heterogeneous software components. Concurrency Comput: Pract. Exper. **14** (2002) 1009–1034

Parallel Hybrid Sparse Solvers Through Flexible Incomplete Cholesky Preconditioning[⋆]

Keita Teranishi and Padma Raghavan

Department of Computer Science and Engineering
The Pennsylvania State University
111 IST Bldg., University Park, PA 16802, USA
{teranish,raghavan}@cse.psu.edu

Abstract. We consider parallel preconditioning schemes to accelerate the convergence of Conjugate Gradients (CG) for sparse linear system solution. We develop methods for constructing and applying preconditioners on multiprocessors using incomplete factorizations with selective inversion for improved latency-tolerance. We provide empirical results on the efficiency, scalability and quality of our preconditioners for sparse matrices from model grids and some problems from practical applications. Our results indicate that our preconditioners enable more robust sparse linear system solution.

1 Introduction

Consider the solution of a sparse linear system $Ax = b$ on a distributed-memory multiprocessor. When A is symmetric positive definite, preconditioned Conjugate Gradients (CG) [5,15] can be used to solve the system. Although CG has scalable memory requirements, its overall performance depends to a large extent on the scalability and effectiveness of the preconditioner. A general purpose preconditioner can be based on incomplete Cholesky with drop thresholds (ICT) [19] to compute $\hat{L}$ such that $\hat{L}\hat{L}^T \approx A$. ICT preconditioning is typically the method of choice on uniprocessors, but its scalable parallel implementation poses many challenges. A major aspect concerns the latency-tolerant application of the preconditioner at every CG iteration; we solve this problem using 'Selective Inversion' (SI) [12,14,17] to replace inefficient distributed triangular solution by effective matrix-vector multiplication. We now report on the overall performance including parallel preconditioner construction and its application. We provide a brief overview of our incomplete Cholesky preconditioning with SI in Section 2. We provide some empirical results on the performance of our schemes in Section 3 followed by the conclusions in Section 4.

2 Parallel Incomplete Cholesky with Selective Inversion (ICT-SI)

Our parallel incomplete Cholesky with SI uses many of the ideas from parallel sparse direct solution. We start with a good fill-reducing strategy such as minimum-degree and

⋆ This work was supported in part by the National Science Foundation through grants ACI-0102537, EIA-0221916, and DMR-0205232.

nested dissection [3,7]; the latter also helps provide a natural data partitioning for the parallel implementations. We then compute an approximate $\hat{L}$ corresponding to the true factor L for the given ordering.

The parallel factorization and triangular solution of $\hat{L}$ is guided by the traversal of the elimination tree [9] in conjunction with supernodes [10,11]. The elimination tree represents the data dependency between columns during factorization; a supernodal tree is a compact version of this tree. Chains in the elimination tree represent a set of consecutive columns of the factor with the same nonzero structure. In a supernodal tree these chains are coalesced into a single supernode. During factorization, dense matrix techniques can be used for columns in a supernode to improve performance through effective cache-reuse.

Our ICT factorization is based on the left-looking variant of sparse factorization, which has been regarded as a memory-efficient scheme compared to the multifrontal approach [11]. During incomplete factorization some nonzero elements are dropped based on a threshold test. Consequently, the column dependencies and the structure for supernodes derived from the ordered coefficient matrix are no longer exact; the factor comprises sparse columns instead of small dense submatrices. However, that structure still enables managing the implementation of a single scheme that can (through dropping) cover the entire range of fill-in to meet a variety of preconditioning needs.

Our parallel implementation takes advantage of individual subtrees which are rooted at the $\log P$ level of the tree to achieve coarse-grain parallelism as shown in Figure 1 for $P = 4$. Incomplete factorization associated with these subtrees proceeds independently and in parallel in a local-phase of computations. At levels above, each supernode is processed by multiple processors using data-parallel dense/blocked operations. At each such, supernode with computations distributed across multiple processors, columns of the incomplete factor are computed by a fan-in scheme with asynchronous communication. This technique allows each processor to overlap the computation of its local elements with interprocessor communication to achieve greater efficiency.

A major performance limitation concerns the parallel application of such ICT preconditioners. At each CG iteration a parallel triangular solution is required. However, when parallel substitution is used for such triangular solution, the performance can degrade considerably due to the relatively large latencies of interprocessor communication. More precisely, the triangular solution at a supernode a involves:

$$\begin{bmatrix} {L^a}_{11} \\ {L^a}_{21} \end{bmatrix} \begin{bmatrix} {x^a}_1 \end{bmatrix} = \begin{bmatrix} {b^a}_1 \\ {b^a}_2 \end{bmatrix} .$$

The submatrices ${L^a}_{11}$ and ${L^a}_{21}$ ($\hat{L}^a_{11}$ and $\hat{L}^a_{21}$ for incomplete factor) are portions of the factor associated with supernode a; ${L^a}_{11}$ is lower-triangular. Parallel substitution is performed to obtain ${x^a}_1$ using ${L^a}_{11}$ and ${b^a}_1$; next ${b^a}_2$ is updated as $b_2 - L_{22}x_1$ and used in an ancestor supernode.

The SI scheme [12,14,17] overcomes the latency problem of substitution through parallel matrix inversion of ${L^a}_{11}$ in the distributed supernodes. Thus, parallel substitution is replaced by the sparse matrix vector multiplication $x_1 \leftarrow L_{11}^{-1}b_1$. This incurs extra computational cost during preconditioner construction, but the improvements in applying

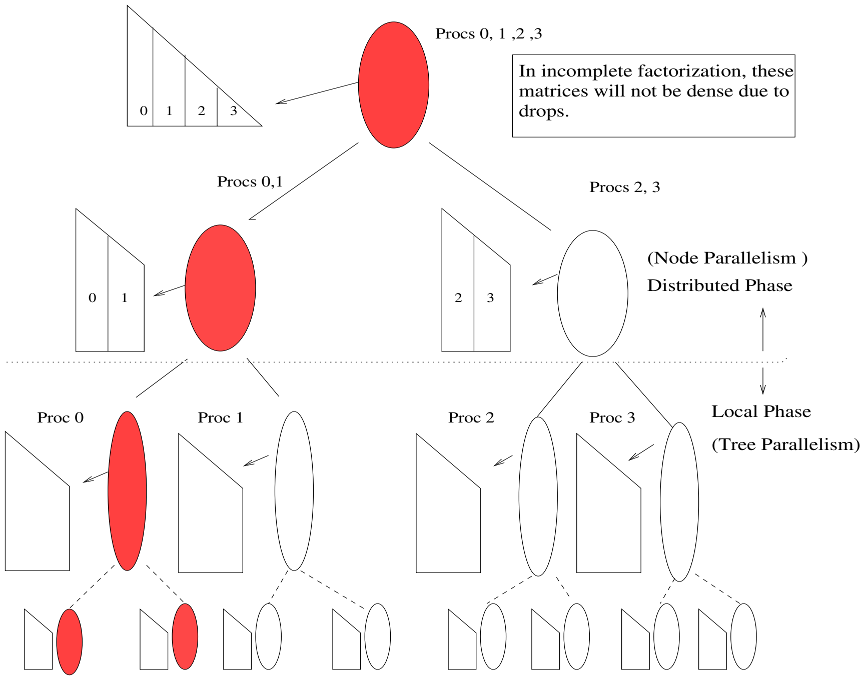


Fig. 1. Parallel ICT factorization on 4 processors using a supernodal tree

the preconditioner can be substantial [14,17]. Our implementation of ICT-SI is based on an incomplete version of the parallel direct solver DSCPACK [13].

3 Empirical Results

We now provide some preliminary results on the performance of our parallel ICT-SI. Additionally, we include for purposes of comparison, results on the performance of level-0 incomplete Cholesky (ICF(0)) [6], and sparse approximate inverse (Parasails) [2] preconditioners. Our experiments were performed on a cluster with 2.4 Ghz Intel Xeon processors and a Myrinet interconnect using parallel CG implementation in the PETSc package [1].

We first report scaled performance when the problem size is scaled with the processors to keep the arithmetic cost of factorization fixed per processor. We used a series of sparse matrices from model grids as shown in Table 1. Figure 2 shows the total time for a single solution and for a sequence of ten right-hand side vectors. The number of iterations, a measure of effectiveness of the preconditioner, is shown in the left half of Figure 3. ICT-SI typically results in lower iterations compared to other methods, but the cost of preconditioner construction is relatively high. We conjecture that the scheme is more appropriate when the cost of preconditioner construction can be amortized over a sequence of solutions for different right hand side vectors (as shown in Figure 2 for a sequence of 10 solutions).

Table 1. Description of sparse test matrices from a model 2 dimensional finite-difference formulation; K is the grid size, N is the matrix dimension, $|A|$ is the number of nonzeroes in the matrix, and $|L|$ is number of nonzeroes in L where $A = LL^T$

Processors (P)	K	N	$\|A\|$	$\|L\|$
			(10^6)	(10^6)
1	200	40000	0.119	1.092
2	250	62500	0.187	1.819
4	312	97344	0.291	2.976
8	392	153664	0.469	4.959
16	490	240110	0.326	3.399
32	618	386884	1.159	13.774

Table 2. Description of sparse matrices. N is the matrix dimension and $|A|$ is the number of nonzeroes in the matrix

Matrix	N	$\|A\|$	Description
augustus5	134,144	645,028	Diffusion equation from 3D mesh
engine	143,571	2,424,822	Engine head, Linear tetrahedral elements
augustus7	1,060,864	9,313,876	Diffusion equation from 3D mesh

We next report on the performance of ICT-SI on three matrices from practical applications, described in Table 2. The results for these three matrices are summarized in Table 3. These results indicate that ICT-SI constructs good quality preconditioners that can significantly reduce the number of CG iterations. However, preconditioner construction costs are still relatively large. When the preconditioner is used for a sequence of ten solutions, ICT-SI is the best performing method for all instances. For **augustus7**, the largest matrix in our collection, we show solution times for one and ten right-hand side vectors using 4–64 processors in Figure 3. These plots indicate that ICT-SI is an effective solver when the costs of preconditioner construction can be amortized over several solutions.

4 Conclusions

We have developed a parallel incomplete Cholesky preconditioner (ICT-SI) which can accommodate the full range of fill-in to meet the preconditioning needs of a variety of applications. Preconditioner construction in ICT-SI relies largely on the techniques derived from efficient parallel sparse direct solvers. Efficient preconditioner application is achieved by using ‘selective inversion’ techniques. The performance of ICT-SI on the matrices from model two-dimensional grid problems compares well with that of sparse approximate inverse preconditioners. However, the cost of preconditioner construction dominates the overall cost of a single solution especially on larger numbers of processors. The preconditioners produced by ICT-SI are quite effective when systems from practical applications are considered. We expect ICT-SI to be useful in applications that produce

Table 3. Performance of parallel preconditioners on three large sparse systems from practical applications. the column labeled ‘Mem’ indicates memory required as multiple of of that for the coefficient matrix. DNC indicates that the method does not converge within 2000 iterations. The best performing instance is shown in boldface

Method	Number of Processors									
	1		2		4		8		16	
	Time	Its	Time	Its	Time	Its	Time	Its	Time	Its
augustus5	Matrix size: 134,144 Nonzeroes: 645,028									
IC(0)	119.3	593	125.7	846	72.5	595	36.5	648	24.7	633
Parasails	28.1	306	24.4	305	18.7	305	9.2	317	**5.1**	306
ICT-SI	**16.2**	**62**	**12.1**	**63**	**11.0**	**69**	**8.2**	**74**	6.2	**79**
	10 right-hand side vectors									
IC(0)	698.5	5925	1018.7	8623	474.3	5924	241.6	6485	152.9	6309
Parasails	217.4	3324	196.3	3342	145.5	3340	66.92	3335	36.8	3344
ICT-SI	**65.27**	624	**44.90**	**630**	**35.26**	**692**	**24.0**	**740**	**20.1**	**740**
engine	Matrix size: 143,571 Nonzeroes: 2,424,822									
IC(0)	239.9	1626	168.5	1707	114.4	1576	99.5	1643	73.8	1699
Parasails	125.9	1036	89.8	1034	62.89	1037	**44.9**	1036	**33.4**	1036
ICT-SI	**48.1**	**282**	**35.2**	**252**	**35.1**	**287**	45.2	**306**	34.2	**308**
	10 right-hand side vectors									
IC(0)	2279.1	16260	1577.3	16958	1104.9	16457	901.8	16337	659.7	17014
Parasails	1166.3	11281	837.6	11329	573.1	11247	408.0	11284	288.2	11277
ICT-SI	**480.1**	**3321**	**240.4**	**2591**	**179.6**	**2862**	**165.9**	**3047**	**132.4**	**3095**
	4		8		16		32		64	
augustus7	Matrix size: 1,060,864 Nonzeroes: 9,313,876									
IC(0)	1569.1	1508	633.0	1238	396.2	1367	193.8	1363	118.7	1377
Parasails	**378.8**	692	**186.7**	692	**103.1**	692	**49.1**	691	**25.9**	691
ICT-SI	473.2	**145**	539.9	**149**	372.9	**157**	275.5	**167**	135.4	**194**
	10 right-hand side vectors									
IC(0)	12652.3	15074	1307.5	15866	723.3	15875	359.7	15892	179.8	15819
Parasails	3478.8	6931	1703.6	6924	934.8	6919	585.7	6924	228.8	6911
ICT-SI	**895.7**	**1449**	**802.9**	**1493**	**542.9**	**1591**	**406.5**	**1672**	**245.4**	**1937**

ill-conditioned sparse matrices especially when the preconditioner can be re-used for a sequence of solutions.

We are currently working to reduce the costs of preconditioner construction [16,18] in our parallel ICT-SI scheme by using sparse approximate inversion [2,4,8] instead of explicit inversion within supernodes.

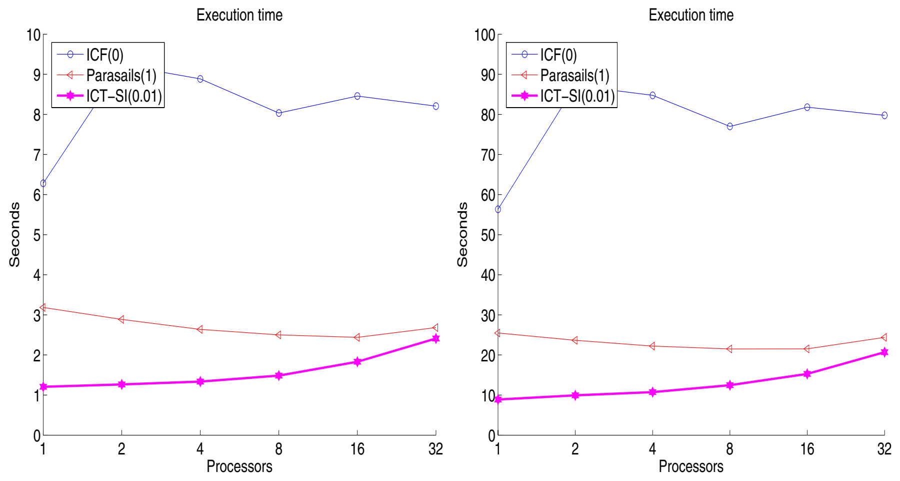


Fig. 2. Total execution time for scaled 2-dimensional grids for a single (left) and ten (right) right-hand side vectors

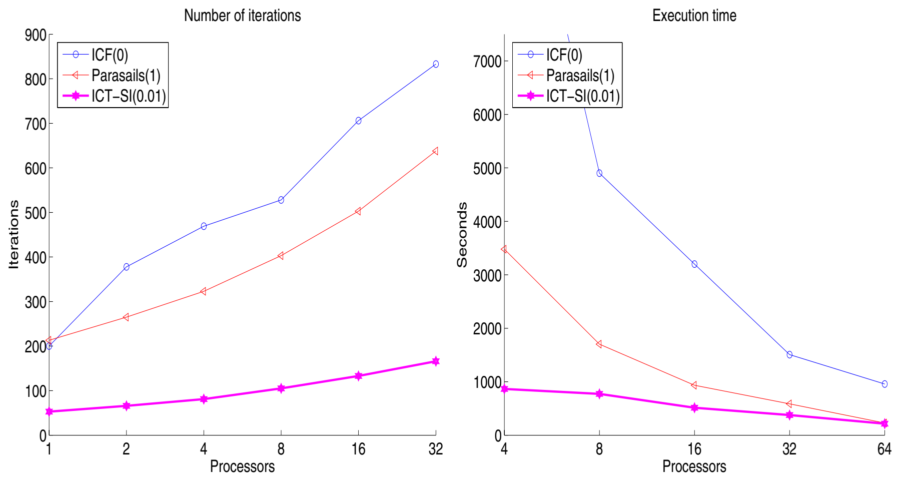


Fig. 3. The number of iterations for convergence of CG for scaled 2-dimensional grids (left) and the execution time for ten solutions for augustus7 (right)

Acknowledgments

We gratefully acknowledge several useful discussions with Barry Smith. We thank Michele Benzi for providing sparse matrices from his collection. We also thank the Mathematics and Computer Sciences Division at the Argonne National Laboratory for allowing us to use their network of workstations.

References

1. BALAY, S., GROPP, W. D., MCINNES, L. C., AND SMITH, B. F. **PETSc** users manual. Tech. Rep. ANL-95/11 - Revision 2.1.1, Argonne National Laboratories, 2002.
2. CHOW, E. Parallel implementation and practical use of sparse approximate inverse preconditioners with a priori sparsity patterns. *Int. J. High Perf. Comput. Apps. 15* (2001), 56–74.
3. GEORGE, A., AND LIU, J. W.-H. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.
4. GROTE, M. J., AND HUCKLE, T. Parallel preconditioning with sparse approximate inverses. *SIAM J. Sci. Comput. 18*, 3 (1997), 838–853.
5. HESTENES, M. R., AND STIEFEL, E. Methods of conjugate gradients for solving linear systems. *National Bureau Standard J. Res. 49* (1952), 409–436.
6. JONES, M., AND PLASSMANN, P. The efficient parallel iterative solution of large sparse linear systems. In *Graph Theory and Sparse Matrix Computations*, A. George, J. R. Gilbert, and J. W. H. Liu, Eds., vol. 56 of *IMA*. Springer-Verlag, 1994, pp. 229–245.
7. KARYPIS, G., AND KUMAR, V. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing 48*, 1 (1998), 71–95.
8. KOLOTILINA, L. Y., AND YEREMIN, A. Y. Factorized sparse approximate inverse preconditionings. I. Theory. *SIAM J. Matrix Anal. Appl. 14*, 1 (1993), 45–58.
9. LIU, J. W.-H. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl. 11*, 1 (1990), 134–172.
10. LIU, J. W.-H., NG, E., AND PEYTON, B. W. On finding supernods for sparse matrix ccomputation. *SIAM J. Matrix Anal. Appl. 14*, 1 (1993), 242–252.
11. NG, E. G., AND PEYTON, B. W. Block sparse Cholesky algorithm on advanced uniprocessor computers. *SIAM J. Sci. Comput. 14* (1993), 1034–1056.
12. RAGHAVAN, P. Efficient parallel triangular solution using selective inversion. *Parallel Processing Letters 9*, 1 (1998), 29–40.
13. RAGHAVAN, P. **DSCPACK**: Domain-separator codes for solving sparse linear systems. Tech. Rep. CSE-02-004, Department of Computer Science and Engineering, The Pennsylvania State University, 2002.
14. RAGHAVAN, P., TERANISHI, K., AND NG, E. G. A latency tolerant hybrid sparse solver using incomplete Cholesky factorization. *Numer. Linear Algebra Appl. 10* (2003), 541–560.
15. SAAD, Y. *Iterative method for sparse linear systems*, second ed. SIAM, Philadelphia, PA, 2003.
16. TERANISHI, K. *Scalable Hybrid Sprase Lienar Solvers*. PhD thesis, Department of Computer Science and Engineering, The Pennsylvania State University, 2004.
17. TERANISHI, K., RAGHAVAN, P., AND NG, E. A new data-mapping scheme for latency-tolerant distributed sparse triangular solution. In *SuperComputing 2002* (2002), pp. 238–247.
18. TERANISHI, K., RAGHAVAN, P., SUN, J., AND MICHARELIS, P. A hybrid preconditioner for sparse systems from thermo-mechanical applications. *Intl. J. Numerical Methods in Engineering* (Submitted).
19. ZLATEV, Z. Use of iterative refinement in the solution of sparse linear systems. *SIAM J. Numer. Anal. 19* (1982), 381–399.

Parallel Heuristics for an On-Line Scientific Database for Efficient Function Approximation[*]

Ivana Veljkovic[1] and Paul E. Plassmann[2]

[1] Department of Computer Science and Engineering
The Pennsylvania State University, University Park, PA 16802, USA
veljkovi@cse.psu.edu

[2] The Bradley Department of Electrical and Computer Engineering
Virginia Tech, Blacksburg, VA 24061, USA
plassmann@vt.edu

Abstract. An effective approach for improving the efficiency of multi-scale combustion simulations is the use of on-line scientific databases. These databases allow for the approximation of computationally expensive functions by archiving previously computed exact values. A sequential software implementation of these database algorithms has proven to be extremely effective in decreasing the running time of complex reacting flow simulations. To enable the use of this approach on parallel computers, in this paper we introduce three heuristics for coordinating the distributed management of the database. We compare the performance of these heuristics on two limiting case test problems. These experiments demonstrate that a hybrid communication strategy offers the best promise for a large-scale, parallel implementation.

1 Introduction

With a new generation of supercomputers, such as the Earth Simulator and Blue Gene, and the availability of commodity parallel computers, such as Beowulf clusters, the computational power available to scientists today is immense. As a result, the physical models that can be simulated on these computers can be extremely complex and sophisticated. Nonetheless, the complete multi-scale simulation of many phenomena can still be computationally prohibitive. For example, in combustion simulations, the inclusion of detailed chemistry with a Computational Fluid Dynamics (CFD) simulation that accurately models turbulence can be intractable. This difficulty appears because a detailed description of the combustion chemistry usually involves tens of chemical species and thousands of highly nonlinear chemical reactions. The numerical integration of these systems of differential equations is often stiff with time-scales that can vary from 10^{-9} seconds to 1 second. Solving the conservation equations for such chemically reacting flows, even for simple, two-dimensional laminar flows, with this kind of chemistry is computationally expensive [1]. For more complex reacting flows, especially those involving turbulence effects, calculations that involve detailed chemistry are typically so

[*] This work was supported by NSF grants CTS-0121573, ACI–9908057, and DGE–9987589 and ACI-0305743.

complex and demanding that they often exceed computational resources of today's most powerful parallel computers.

One approach to address this problem involves the implementation of a specialized database to archive and retrieve accurate approximations to repetitive expensive calculations. That is, instead of solving the equations at every point required by the simulation, one can solve it for some significantly smaller number of points and interpolate these solutions to obtain approximations at nearby points. This approach was originally proposed by Pope for combustion simulations [2]. In his paper Pope proposed to tabulate previously computed function values; when a new function value is required, the set of previously computed values is searched and, if possible, the function is approximated based on these values. A mechanism based on ellipsoids of accuracy was introduced to estimate the error of the function approximations retrieved from the tabulation. In this manner, the tabulation scheme attempts to keep this error within a user-specified tolerance.

This approach also has the potential to be used in other application areas. For example, it could be used in radiative heat transfer calculations involving non-gray radiation models—when the radiative properties have a strong wavelength dependence. With these models, frequent, computationally expensive evaluations of the absorption coefficient as a function of temperature, pressure, and mass fraction of chemical species and wavelength are required. These coefficients could be similarly archived and queried using this database approach. Another potential application is the modeling of biological systems, such as the interaction of viruses, antibodies and cells, via the solution of systems of differential equation. Such systems also lead to extremely stiff equations. We are investigating the potential use of this database approach to improve the efficiency of such calculations.

In previous work we introduced a sequential algorithm that extended Pope's approach in several directions [3]. First, we introduced a dynamic updating of the database based on usage statistics to accommodate non-steady state simulations. Second, we developed a geometric partitioning algorithm that guarantees the retrieval of a database entry if it satisfies the error tolerance for the function approximation. These algorithms are implemented in the software system DOLFA.

Because of the importance of the use of parallel computers to solve large-scale combustion problems, it is extremely desirable to develop a parallel version of this sequential algorithm. In previous work we proposed an a approach for distributing the function database and introduced three communication strategies for addressing load balancing problems in the parallel implementation [4]. In this paper we introduce a detailed formulation of these three communication heuristics and present computational results for two limiting cases that demonstrate the advantage of a proposed hybrid strategy.

The remainder of our paper is structured as follows. In Section 2 we briefly review our sequential algorithm as implemented in the software package DOLFA [3]. In Section 3 we describe the initial parallel implementation of DOLFA as presented in [4]. In this section we also present a detailed formulation of three proposed communication heuristics. Finally, in Section 4 experimental results are presented showing the parallel performance of our three proposed load balancing strategies that illustrates the advantage of a hybrid communication heuristic.

2 The Sequential Scientific Database Algorithm

The function values to be computed, stored, and approximated by the database are n-dimensional functions of m parameters, where both n and m can be large (in the 10's or 100's for typical combustion problems). We denote this function as $F(\theta)$ which can be stored as an n-dimensional vector and where the parameters θ can be represented by an m-dimensional vector. The sequential database algorithm works by storing these function values when they are directly (exactly) computed. If possible, other values are approximated using points stored in the database. In addition to the function values, the database stores other information used to estimate the error based on an Ellipsoid Of Accuracy (EOA) as described in [3].

To illustrate how the sequential database works, when $F(\theta)$ is to be calculated for some new point, we first query the database to check whether this value can be approximated with some previously calculated value. For computational combustion, the parameter space is often know as the *composite space*. The composite space consists of variables that are relevant for the simulation of a chemical reaction, such as temperature, pressure and mass fractions of chemical species.

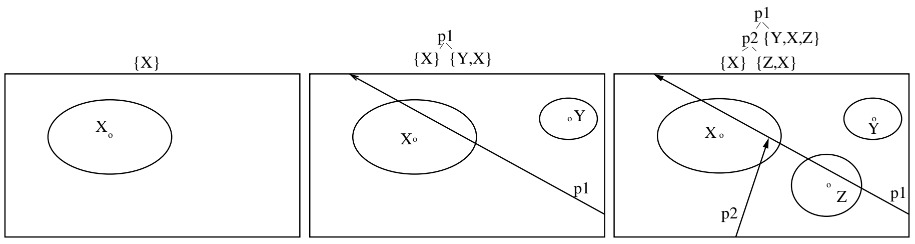


Fig. 1. The generation of a BSP tree and the resulting spatial decomposition of a two-dimensional domain from the insertion of three data points. Arrows illustrate the orientation of the cutting planes and the ovals represent the EOAs. Figure taken from [3]

In DOLFA, the database is organized as a Binary Space Partition (BSP) tree to enable the efficient search of the composite space. Internal nodes in the BSP tree denote planes in the m-dimensional space. Terminal nodes (leafs) represent convex regions determined by the cutting planes on the path from the given terminal node to the root of the BSP tree. This construction is illustrated in Fig. 1. For each region we associate a list of database points whose EOAs intersect the region. In addition, to avoid storing points that are unlikely to be used for approximation again, we employ techniques based on usage statistics that allow us to remove such points from the database.

The software system DOLFA has demonstrated significant speedups for combustion applications. For example, the software was tested with a code that implements a detailed chemistry mechanism in an HCCI piston engine [5]. For this application, when compared to simulation results based only on direct integration insignificant effects on the accuracy of the computed solution were observed.

3 The Parallel Database Algorithm

In developing a parallel version of the database, perhaps the most straightforward approach would be to use the sequential version on each processor, without any interprocessor collaboration in building a distributed database. This approach would work; however, it suffers from two significant drawbacks. First, this approach cannot take advantage of direct calculations done on other processors—hence, we would expect significant redundant function calculations. Second, the memory required to store the database can be significant—without partitioning the database among processors, we cannot take full advantage of the total memory available on a parallel computer. In this section, we present parallel algorithms that address each of these issues.

The importance of addressing the first issue of redundant evaluations becomes obvious if we compare typical times for function evaluation and interprocessor communication. Namely, for the HCCI combustion simulation which was considered earlier in Section 2, one function evaluation takes 0.3 seconds (an ODE system with 40 chemical species). On the other hand, on the Beowulf cluster where our experiments were conducted the time to communicate the computed results (40 double precision values) is approximately 2.7 microseconds. The relative difference in times is significant, and even if we consider all the synchronization issues, it is still important to design a parallel approach that will coordinate direct calculations between processors to minimize the number of redundant direct calculations. Thus, our goal is to design a parallel implementation of DOLFA that minimizes the total number of direct calculations performed on all processors. However, this minimization must be obtained while maintaining good load balancing and while minimizing the total interprocessor communication.

The parallel algorithm consists of two main parts. First, the approach used for building the distributed (global) BSP tree and second, the heuristics developed for managing interprocessor communication and the assignment of direct function evaluations. In next two subsections we will discuss solutions to these two problems.

3.1 Building the Global BSP Tree

As with the sequential case, we typically have no *a priori* knowledge of the distribution of queries within the search space. The global BSP tree has to be built on-line; it will be partitioned into unique subtrees which are assigned to processors as illustrated in Fig. 2. To achieve a well-balanced partitioning of the global BSP tree, we initially accumulate a certain number of points in the database without constructing the BSP tree. The more points we accumulate, the better chance that the distributed BSP tree will partition the space in a well balanced manner, since the partitioning is based on more information about the accessed space. On the other hand, the longer we wait before constructing the global BSP tree, the greater the chance that we will do redundant direct calculations.

We must balance these two issues and attempt to design a strategy that will yield sufficient information about the accessed space as early as possible. A formula based on the average number of points in the database (total number of points divided with number of processors) is used to determine when to build the BSP tree. After a sufficient number of points is accumulated in this initial phase, our goal is to have enough information to construct a BSP tree for the entire search space with reasonably balanced subtrees.

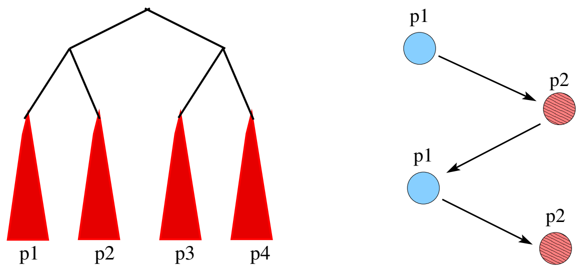


Fig. 2. On the left: an illustration of how the global BSP tree is partitioned among processors—each processor owns a unique subtree of the global tree. On the right: an illustration of the "ping-pong" communication between processors. Figure taken from [4]

These subtrees are computed by recursively decomposing the global search space by introducing cutting planes that roughly equally divide the set of points in each subregion. The resulting subtrees (and their corresponding subregions) are then assigned to individual processors. The choice of cutting planes is limited to coordinate planes chosen by computing the dimension with maximum variance. As we mentioned before, each processor owns its subtree but also has information about the portion of the search space stored on every other processor in the environment. Therefore, the search for a suitable point ϕ to use for approximation for point θ is conducted in a coordinated manner between processors. More precisely, if the query point θ is submitted to processor i but this point belongs to the subtree of processor j, processor i will be able to recognize this fact. A simple model for the computational complexity of this initial phase of the parallel database algorithm was developed in [4]. This is a one-time cost which is not significant.

3.2 Managing Interprocessor Communication

Before each call to the database from the main driver application, all processors must synchronize to be able to participate in global database operations. This synchronization may be very costly if it is conducted often. For example, if the database is called with a single point at a time as is done with the sequential version of DOLFA. Modifying the DOLFA interface by allowing calls that are made with lists of query points instead of one point at a time minimizes the synchronization overhead and enables more efficient interprocessor communication.

Fortunately, in most combustion simulations, updates to the fluid mechanics variables and chemical species mass fractions happen independently of one another on the mesh subdomain assigned to each processor. More precisely, the chemical reaction at point P at time t does not affect the chemical reaction at point Q at time t. Therefore, we can pack the points for which we want to know function values into a list and submit it to the database. The output is a list of function values at points given in the input list.

Based on our assumption that the function evaluation is relatively expensive, we want to design a searching scheme that will minimize the number of direct evaluations that must be performed. Whenever there is a possibility that another processor may be able to approximate the function value at a given point, we would like to communicate that query to that processor for processing. This means that, for a given query point X submitted to processor $p1$, we have one of two possibilities. First, when point X belongs to a subtree $T1$ of $p1$ (see Fig. 2-left), we simply proceed with the sequential algorithm. Second, the more challenging case is when the point X belongs to a subtree of some other processor, say $p2$, $X \in T2$. As $p1$ stores some number of points that belong to processor $p2$ (for example, from the first phase of computation when BSP tree is not yet built) $p1$ may still be able to approximate this query. If it cannot, the processor sends the point to processor $p2$. If $p2$ can approximate it, it sends back the result. But, if $p2$ also cannot satisfy the query, the open question is to determine which processor should conduct the direct integration.

The biggest issue to consider in answering this question is to determine the effect on load balancing between processors. It may happen that, for a particular time step, all the points for which we want to calculate function values belong to a small subregion of the accessed space. This effect is common in many combustion simulations. If the load imbalance is large, we may loose the performance advantages gained by minimizing the number of redundant direct calculations. Thus, the answer to the question: “Should $p1$ or $p2$ conduct the direct calculation?” proves to be a crucial part of the parallel implementation of DOLFA.

We introduce the following notation to describe the three heuristics proposed in this paper. We denote by $\mathcal{L}^{(i)}$ the list of points submitted to processor p_i. This list is decomposed in sublists $\mathcal{L}^{(i)} = \bigcup_{j=1}^{n} \mathcal{L}_j^{(i)}$ where $\mathcal{L}_j^{(i)}$ is the portion of the list that belongs to subtree T_j of processor p_j, $\mathcal{L}_j^{(i)} = \mathcal{L}^{(i)} \bigcap T_j$. So every processor p_i will have three different types of list to deal with:

- $\mathcal{B}^{(i)} = \mathcal{L}_i^{(i)}$ – the list of points that belong to the subtree of p_i. This is the list of the points that processor p_i has to calculate.
- $\mathcal{N}^{(i)} = \bigcup_{j \neq i} \mathcal{L}_j^{(i)}$ – the list of points that belong to the subtrees of other processors. This is the list of the points that processor p_i will send off to other processors.
- $\mathcal{E}^{(i)} = \bigcup_{j \neq i} \mathcal{L}_i^{(j)}$ – the list of points that p_i received from other processors since they belong to its subtree T_i.

With $\mathcal{C}^{(i)}$ we denote the list of points for which processor p_i does the direct calculations.

The open question remains: “If X belongs to $\mathcal{L}_2^{(1)}$ – it was submitted to p_1 for calculation but it belongs to a subtree T_2 of processor p_2 – which processor will calculate F(X)?". We compare three approaches for solving this problem as described below.

Approach 1. The owner of the subtree ($p2$ in the above example) does the direct integration. In this approach, processor p_i does the direct calculations for all the points in the lists $\mathcal{B}^{(i)}$ and $\mathcal{E}^{(i)}$, $\mathcal{C}^{(i)} = \mathcal{B}^{(i)} \bigcup \mathcal{E}^{(i)}$. This approach leads to maximum retrieval rates but introduces a significant load imbalance. Because problems in combustion chemistry usually include turbulence and nonhomogeneous media, this approach can result in significant load imbalance. However, as this approach should minimize the number of direct calculations, it can be used as a benchmark determining a lower bound for the overall number of direct calculations. This lower bound can be used to compare with other approaches to estimate the number of redundant function evaluations.

Approach 2. The processor to which the query was submitted ($p1$ in the above example) does the direct calculations. In this approach, processor p_i does the direct calculations for all the points in the list $\mathcal{B}^{(i)}$ and for some of the points in the list $\mathcal{N}^{(i)}$, where $\mathcal{C}^{(i)} = \mathcal{B}^{(i)} \bigcup \bar{\mathcal{N}}^{(i)}$ (bar denotes a subset of the list). This approach may mitigate the load imbalance, but it will introduce a communication overhead as we have additional two-way communication. This communication would consist of processor $p2$ letting processor $p1$ know that it cannot approximate $F(X)$ and processor $p1$ returning to $p2$ the calculated $F(X)$ together with some other items needed to calculate the ellipsoid of accuracy discussed in subsection 2. We describe this communication pattern as a "ping-pong," as illustrated in the right of Fig. 2. A problem with this approach is a significant reduction in retrieval rates. For example, lists on processors $p3$ and $p4$ may contain points that are close to X and that could be approximated with $F(X)$. In approach 1, we would have only one direct calculation for $F(X)$. In this approach, since $p2$ will send off point X back to original processors to calculate it (in this case, $p1$, $p3$ and $p4$), we would have these three processors redundantly calculate $F(X)$.

Hybrid Approach. We consider a hybrid version combining the best of the two previous approaches. In this approach, processor p_i does the direct calculations for all the points in the list $\mathcal{B}^{(i)}$ and for some of the points in the lists $\mathcal{N}^{(i)}$ and $\mathcal{E}^{(i)}$, where $\mathcal{C}^{(i)} = \mathcal{B}^{(i)} \bigcup \bar{\mathcal{N}}^{(i)} \bigcup \bar{\mathcal{E}}^{(i)}$ (bar denotes a subset of the list). Processor p_i decides whether to calculate $F(X)$ for some $X \in \mathcal{E}^{(i)}$ based on the average number of direct retrievals in the previous iteration and additional metric parameters. In particular, processor p_i can predict how many calculations will it have in the $\mathcal{B}^{(i)}$ and $\mathcal{N}^{(i)}$ list—denote this number with $avgLocal$. The number of direct calculations can vary substantially in the course of computation; therefore, our prediction has to be local and we calculate it based on the two previous iterations. Processor p_i also knows the average number of direct retrievals (over all processors) in the previous iteration—denote it with $avgDirect$. Then, processor p_i will do no more than $avgExtra = avgDirect - avgLocal$ direct calculations on the $\mathcal{E}^{(i)}$ list.
For the rest of the points in the $\mathcal{E}^{(i)}$ list, p_i will send them back to the original processors to calculate them, using the "ping-pong" communication model. Based on this decision, $F(X)$ is either computed on this processor or sent to the original processor for evaluation.

4 Experimental Results

We have designed a test application that mimics typical combustion simulations to test the performance of our parallel implementation. This application mimics a two-dimensional laminar reacting flow. The model parameters are the following: the number of time steps is 20, the mesh size is 50×50 and the vector size (size of query point) is 10. The user-defined tolerance is 0.01 and it takes roughly 0.1 second to perform one direct evaluation. Our experiments were run on a parallel Beowulf cluster with 81 AMD MP2000+ dual processor nodes, 1 GB memory per node, and a high-speed Dolphin interconnection network.

We ran two limiting case experiments which we refer to as Experiments 1 and 2. For Experiment 1, each processor gets a different domain on which to calculate. Thus,

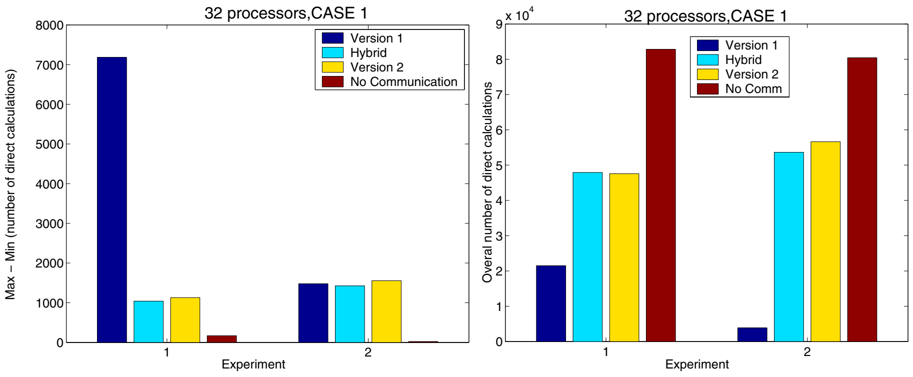


Fig. 3. Left: Difference between the maximum and minimum number of direct calculations across the parallel environment. Load imbalance is obvious for algorithm version 1. Right: Overall (summed over all processors) number of direct calculations across the parallel environment The number of redundant calculations is very high for the case when we have no communication between processors

for this case, the number of direct calculations should be roughly a linear function of the number of processors. For Experiment 2, each processor gets the same domain. For this case the number of direct calculations should be roughly constant function of the number of processors. The objective is to observe how will the load balancing and communication patterns behave in these extreme cases.

We first compared the load balancing for each of the communication heuristics described in the preceding section. Load balancing was measured by computing the difference between the maximum and minimum number of direct calculations across all processors. If this relative difference is near zero, this implies that roughly every processor conducted the same amount of work over the entire calculation (however, there may still be load imbalance at individual time-steps).

On the left of Fig. 3, we present the results for Experiments 1 and 2 run on 32 processors. As can be seen, version 1 has the largest load imbalance while the hybrid approach achieves the best load balance. On the right of Fig. 3 we compare the total number of direct calculations (summed over all processors in the environment) for Experiments 1 and 2 run on 32 processors. As expected, we observe that the no-communication implementation has a significant number of redundant calculations compared to any version of the parallel algorithm.

In the next series of simulations, we keep the load constant as we increase the number of processors. In Fig. 4 we compare the average overhead as a function of the number of processors used for the four strategies (version 1, hybrid approach, version 2 and no-communication) for Experiment 1 (left) and Experiment 2 (right). The overhead is calculated as the difference between the overall time spent performing database operations and time to do the direct calculations. For the Experiment 1, where the overall number of direct calculations is roughly a linear function of number of processors, the computational overhead is unacceptable for version 1 as it increases as the number of

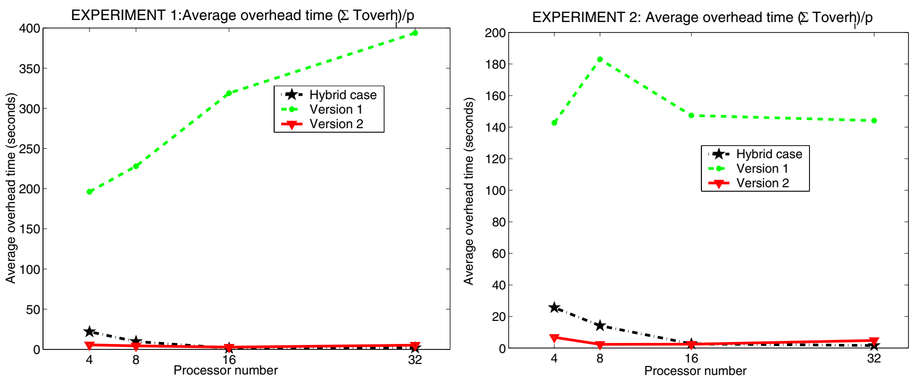


Fig. 4. Average time overhead as a function of number of processors for our 4 schemes. Version 1 has a prohibitive overhead

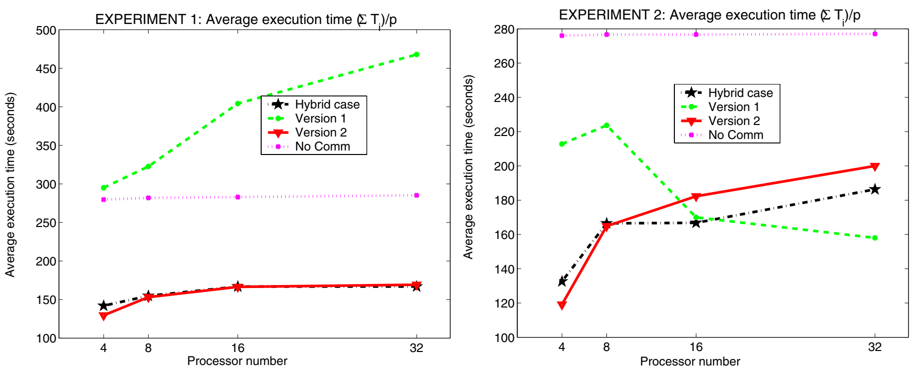


Fig. 5. Average execution time as a function of number of processors for our 4 schemes. Hybrid version and version 2 exhibit scalability. However, version 1 does a better job in minimizing the number of direct calculations

processors increase. Therefore, even though we observe on the right of Fig. 3 that version 1 minimizes the overall number of direct calculations better than any other scheme, the load imbalance is high (Fig. 3,left) and this scheme fails to meet our goals.

However, for Experiment 2, the overhead for version 1 is roughly constant which implies that this scheme appears to be more scalable. We expect this approach will work well for large numbers of processors when we know that the number of direct calculations is a slowly growing function of number of processors. Because we expect that real-time applications will be more like the extreme situation described in Experiment 1, version 1 is in our opinion not a feasible parallel scheme. But as it has some limiting properties, it can be used for comparison to other schemes that we design. We also notice that the overhead is roughly constant for the hybrid approach.

In Fig. 5 we compare the average execution time as a function of the number of processors for the four strategies with Experiment 1 shown on the left and Experiment

2 on the right. If we compare Figs. 4 and 5, we note that the overhead time is actually dominating the running time for version 1. This effect also demonstrates that version 1 is not a practical scheme for communication for our on-line database. On the other hand, we notice that the average execution time for version 2 and the hybrid scheme becomes constant for Experiment 1 and almost constant for Experiment 2 as we increase the number of processors. This suggest that these schemes should be scalable for larger problems and numbers of processors. However, the hybrid version has an advantage over version 2 since it performs less redundant direct calculations.

5 Conclusions

We showed in our previous work that sequential database system DOLFA can significantly reduce the execution time for large, multi-scale simulations that involve frequent expensive function evaluations and we introduced its parallel extension. In this paper, we give a detailed description of the parallel algorithms that are crucial for the implementation and conduct experiments that demonstrate the performance of the proposed schemes. Our approach is based on a partitioning of the search space and maintaining a global BSP tree which can be searched on each processor in a manner analogous to the sequential database algorithm. Several heuristics have been introduced which aim to balance the computation of points which are not resolved locally. In the future work, we plan to test our parallel implementation with an application that models reacting flow with complex chemistry using a high-order, Direct Numerical Simulation (DNS) fluid simulation. In addition, we plan to conduct experiments on larger numbers of processors and further examine the scalability of these approaches.

References

1. U. Maas and S.B. Pope. Laminar flame calculations using simplified chemical kinetics based on intrinsic low-dimensional manifolds. Twenty-Fifth Symposium (International) Combustion/The Combustion Institute, pages 1349-1356, 1994.
2. S.B. Pope. Computationally efficient implementation of combustion chemistry using *in situ* adaptive tabulation. Combustion Theory Modeling, Vol 1, pages 41-63, 1997.
3. I. Veljkovic, P.E. Plassmann and D.C. Haworth. A scientific on-line database for efficient function approximation. Computational Science and Its Applications—ICCSA 2003, The Springer Verlag Lecture Notes in Computer Science (LNCS 2667) series, Part I, pages 643–653, 2003.
4. I. Veljkovic, P.E. Plassmann and D.C. Haworth. A parallel implementation of a scientific on-line database for efficient function approximation. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications—PDPTA 2004, Vol 1, pages 24-29, 2004.
5. M. Embouazza, D.C. Haworth and N. Darabiha. Implementation of detailed chemical mechanisms into multidimensional CFD using *in situ* adaptive tabulation: Application to HCCI engines. Journal of Fuels and Lubricants 111, pages 1544-1559, 2002.
6. S.B. Pope. ISAT-CK User manual (Version 3.0). October 2000.

Software Engineering and Problem Solving Environments for Scientific Computing: An Introduction

Organizers: Jose C. Cunha[1] and Omer F. Rana[2]

[1] Universidade Nova de Lisboa, Portugal
jcc@di.fct.unl.pt
[2] Cardiff University, UK
o.f.rana@cs.cardiff.ac.uk

As computational infrastructure becomes more powerful and complex, there is a greater need to provide tools to support the scientific computing community to make better use of such infrastructure. The abscence of such tools is likely to lead to users not taking full advantage of newer functionality made available in computational infrastructure. It is also unlikely that a user (a scientist or a systems administrator) will know the full range of options available – or how a combination of configuration options from elements within the infrastructure (such as compute or data servers) can be usefully specified. This issue has two aspects: (1) allowing such functionality to be made available in tools without requiring the user to know about the functionality – an example includes the ability to support service discovery. In this case, the user does not really need to know where a particular service instance resides, only that a particular type of service is available for use within their application at a given point in time. (2) increasing the reliance of the user on tools – an example includes the ability to automatically compose a set of services based on the requirement outlined by a user. In this case, a user must place greater trust in the infrastructure to deliver a particular result. This aspect has not been fully realised within existing infrastructure, and remains an active research area. It involves both the need to better integrate functionality made available in computational infrastructure, but also requires a "culture change", in the way that users perceive and make use of computational infrastructure.

The last decade has also seen an unprecendented focus on making computational resources sharable (parallel machines and clusters, and data repositories) across national boundaries. Significantly, the emergence of Computational Grids in the last few years, and the tools to support scientific users on such Grids (sometimes referred to as e-Science) provides new opportunities for the scientific community to undertake collaborative, and multi-disciplinary research. Often tools for supporting application scientists have been developed to support a particular community (Astrophysics, Biosciences, etc), a common perspective on the use of these tools and making them more generic is often missing.

On the other hand, the software engineering community in computer science often aims to develop approaches which are more generic, and can be more widely deployable. Research into Problem-Solving Environments (PSEs) is also an active and expanding field, with the potential for a wide impact on how computational resources can be harnessed in a more effective manner. Ideally, PSEs should provide software tools and expert assistance to a user, and serve as an easy-to-use interface to support the development and deployment of scientific applications, thereby allowing the rapid prototyping of ideas, and support for undertaking analysis on experimental data.

The aim of this minisymposium was to bring together experts who have experience of developing (implementing and deploying) software tools to support application scientists, and those who make use of these tools. A particular aim is to highlight lessons learned – and, more importantly, identify good (and bad) approaches. The paper by Cannataro et al. describes different elements of a PSE, and why such elements are useful to support. A key aspect is a focus on Grid-based PSEs. Schreiber describes the TENT system – a PSE for use in Aerospace Engineering applications. The author describes how new components may be integrated within the PSE, and issues that arise in this process. The paper by Overeinder and Brazier, and by Jost et al. describe the inclusion of artificial intelligence based approaches within PSEs and the computational infrastructure. Overeinder and Brazier report on the “AgentScape" project, which provides distributed Operating Systems-like functionality for supporting computational services. The elements of the AgentScape system are outlined – with particular reference to the problem solving capability of such infrastructure. Jost et al. report on an Expert System-based approach for supporting a user in executing their application on an existing infrastructure – based on dynamic performance data, and on the static structure of the application.

A General Architecture for Grid-Based PSE Toolkits

Mario Cannataro[1], Carmela Comito[2], Antonio Congiusta[2], Gianluigi Folino[3], Carlo Mastroianni[3], Andrea Pugliese[2,3], Giandomenico Spezzano[3], Domenico Talia[2], and Pierangelo Veltri[1]

[1] Università "Magna Græcia" di Catanzaro
{cannataro,veltri}@unicz.it
[2] DEIS, Università della Calabria
{apugliese,comito,congiusta,talia}@si.deis.unical.it
[3] ICAR-CNR
{folino,mastroianni,spezzano}@icar.cnr.it

Abstract. A PSE Toolkit can be defined as a group of technologies within a software architecture that can build multiple PSEs. We designed a general architecture for the implementation of PSE toolkits on Grids. The paper presents the rationale of the proposed architecture, its components and structure.

1 Introduction

Problem Solving Environments (PSEs) have been investigated over the past 30 years. In the pioneering work "A system for interactive mathematics", Culler and Fried in 1963 initiated to investigate automatic software systems for solving mathematical problems with computers, by focusing primarily on applications issues instead of programming issues. At that time, and later for many years, the term "applications" indicated scientific and engineering applications that were generally solved using mathematical solvers or scientific algorithms using vectors and matrices. More recently, PSE for industry, commercial, and business applications are gaining popularity.

Nowadays, there is not a precise definition of what a PSE is. The following well-known definition was given by Gallopoulos, Houstis, and Rice [7]:

"*A PSE is a computer system that provides all the computational features necessary to solve a target class of problems. [...] PSEs use the language of the target class of problems.*"

Moreover, they tried to specify the main components of a PSE by defining the equation

PSE = Natural Language + Solvers + Intelligence + Software Bus

where

– *Natural Language* is specific for the application domain and natural for domain experts;
– *Solvers* are software components that do the computational work and represent the basic elements upon which a PSE is built;

- *Intelligence* is useful for resource location, input assistance (with recommender interfaces), scheduling, and analysis of results;
- *Software Bus* is the software infrastructure that supports the PSE and its use.

Other definitions that partially agree with the above definition have been given in the last five years. According to Walker et al. [12],

> "*a PSE is a complete, integrated computing environment for composing, compiling, and running applications in a specific area*."

whereas Schuchardt and his co-authors defined PSEs as [11]

> "*problem-oriented computing environments that support the entire assortment of scientific computational problem-solving activities ranging from problem formulation to algorithm selection to simulation execution to solution visualization. PSEs link a heterogeneous mix of resources including people, computers, data, and information within a seamless environment to solve a problem*."

Finally, in 2003, Cunha defined a PSE as [6]

> "*an integrated environment for solving a class of related problems in an application domain; easy to use by the end-user; based on state-of-the-art algorithms. It must provide support for problem specification, resource management, execution services*."

The existence of several different definitions demonstrates that the term PSE is perhaps too generic and not completely investigated for reaching a full consensus in the scientific community.

The main motivation for developing PSEs is that they provide software tools and expert assistance to the computational scientist in a user-friendly environment, thus allowing for more rapid prototyping of ideas and higher research productivity. A PSE provides users with an interface to high-performance computing resources, relieving them from hardware/software details and letting them concentrate on applications; *collaboration*, *visualization*, and *knowledge* are the three main aspects that distinguish a PSE from a mere interface.

As specified in the PSE definitions given above, a PSE is typically aimed at a particular computing domain. An advancement of the PSE concept is the PSE Toolkit concept. A PSE Toolkit can be defined as a group of technologies within a software architecture that can build multiple PSEs. A PSE Toolkit allows for building meta-applications from preexisting modules, in order to meet the needs of specific solutions. It can support the construction of problem solving environments on different domains allowing designers to develop multidisciplinary PSEs.

PSEs can benefit from advancements in hardware/software solutions achieved in parallel and distributed systems and tools. One of the most interesting models in the area of parallel and distributed computing is the Grid. This paper presents a general architecture for Grid-based PSE Toolkits. Section 2 describes the issues to be faced in designing Grid-based PSE Toolkits and reviews some related work. Section 3 presents the reference architecture, and Section 4 concludes the paper.

2 Grid-Based PSE Toolkits

As said before, PSEs are typically designed with a specific application domain in mind. This approach simplifies the designer task and generally produces an environment that is particularly tailored for a particular application class. On the other hand, this approach does limit portability of solutions. That is, the resulting PSE cannot be generally used in different application domains without re-designing and re-implementing most or all of the environment. PSE portability, extensibility, and flexibility can be provided using the PSE Toolkit model.

PSE Toolkit designers provide the most important components for building up PSEs and a way to compose them when a specific PSE must be developed. Therefore, a PSE Toolkit allows for developing domain-specific PSEs, thus creating meta-applications from pre-existing modules on different domains.

PSE users, which often expect to be interfaced with a tightly integrated environment, must have transparent access to disperse and de-coupled components and resources. Dealing with distributed environments is another main issue in PSEs design and use. Distributed and parallel computing systems are used for running PSEs to get high performance and use distributed data, machines, or software modules.

The Grid is a high-performance infrastructure that combines parallel and distributed computing systems. Its main goal is scalable and secure resource sharing and coordinated problem solving in “dynamic, multi-institutional virtual organizations”. The role of the Grid is fundamental, since it potentially provides an enormous amount of dispersed hardware and software resources; such resources can be transparently accessed by a PSE Toolkit. In Grid environments, instruments can be connected to computing, data, and collaboration environments and all of these can be coordinated for simultaneous operation. Moreover, vast quantities of data can be received from instruments and simulations, and catalogued, archived, and published. The Grid can thus provide a high-performance infrastructure for running PSEs and, at the same time, a valuable source of resources that can be integrated in PSEs and PSE Toolkits; therefore, Grid-aware PSEs can search and use dispersed high performance computing, networking, and data resources.

Through the combination of PSE Toolkit issues and the exploitation of Grid features, we have the possibility to design Grid-based PSE Toolkits. Some main issues to deal with in designing a Grid-based PSE Toolkit are:

- common and domain-specific components identification;
- understanding and evaluation global properties;
- component integration;
- distribution of resources and components;
- component middleware, technology, and infrastructures;
- adaptivity and heterogeneity;
- standardization;
- efficiency.

Common components can be designed and implemented in a PSE Toolkit and they can be used when a specific PSE must be developed. Design of application-bound components and their interfaces must be considered in the Toolkit, but its implementation will change depending on the PSE in which they will be included. Most of the PSE Toolkit components are common. Specific components are:

– components that define goals, resources, actors of the application domain;
– ontology of the domain components;
– domain knowledge;
– performance history;
– the interface between the application domain and the PSE Toolkit.

Grid-based PSEs may succeed in meeting more complex applications requirements both in terms of performance and solution complexity. They integrate heterogeneous components into an environment that provides transparent access to distributed resources, collaborative modeling and simulation, and advanced interfaces.

2.1 Related Work

In this section we give a short review of some existing Grid-based PSE Toolkits, so as to illustrate their main characteristics and the different approaches adopted.

The WebFlow toolkit [1] is a Web-based three-tier system where high-performance services are implemented on the Grid using the Globus toolkit [8]. WebFlow provides a job broker to Globus, while Globus takes responsibility of actual resource allocation. The WebFlow front-end allows specifying user's task in the form of an Abstract Task Descriptor (ATD). The ATD is constructed recursively and can comprise an arbitrary number of subtasks. The lowest level, or atomic, task corresponds to the atomic operation in the middle tier, such as instantiation of an object, or establishing interactions between two objects through event binding. A mesh of CORBA-based servers gives the WebFlow middle tier; one of these servers, the gatekeeper server, facilitates a secure access to the system. The middle-tier services provide the means to control the life cycles of modules and to establish communication channels between them. Services provided by the middle tier include methods for submitting and controlling jobs, manipulating files, providing access to databases and mass-storage, and querying the status of the system and users' applications. WebFlow applications range from land management systems to quantum simulations and gateway seamless access.

GridPort [9] is an open architecture providing a collection of services, scripts and tools that allow developers to connect Web-based interfaces to the computational Grid behind the scenes. The scripts and tools provide consistent interfaces between the underlying infrastructure and security, and are based on Grid technologies such as Globus and standard Web technologies such as CGI and Perl. GridPort is designed so that multiple application portals share the same installation of GridPort, and inherit connectivity to the computational Grid that includes interactive services, data, file, and account management, and share a single accounting and login environment. An interesting application of GridPort is a system called Hot-Page, which provides users with a view of distributed computing resources and allows individual machines to be examined about status, load, etc.; moreover, users can access files and perform routine computational tasks.

The XCAT Grid Science Portal (XCAT-SP) [10] is an implementation of the NCSA Grid Science Portal concept, that is a problem solving environment that allows scientists to program, access and execute distributed applications using Grid resources which are launched and managed by a conventional Web browser and other desktop tools. XCAT-SP is based on the idea of an "active document" which can be thought of as a "notebook" containing pages of text and graphics describing the structure of a particular application and pages of parameterized, executable scripts. These scripts launch and manage an application on the Grid, then its results are dynamically added to the document in the form of data or links to output results and event traces. Notebooks can be "published" and stored in web based archives for others to retrieve and modify. The XCAT Grid Science Portal has been tested with various applications, including the distributed simulation of chemical processes in semiconductor manufacturing and collaboratory support for X-ray crystallographers.

The Cactus Code and Computational Toolkit [3] provides application programmers with a high level set of APIs able to hide features such as the underlying communication and data layers; these layers are implemented in modules that can be chosen at runtime, using the proper available technology for each resource. Cactus core code and toolkits are written in ANSI C, offer parallel I/O, checkpointing and recovery of simulations, and provide users with an execution steering interface. Much of the Cactus architecture is influenced by the vast computing requirements of its main applications, including numerical relativity and astrophysics. These applications, which are being developed and run by large international collaborations, require Terabyte and Teraflop resources, and will provide an ideal test-case for developing Grid computing technologies for simulation applications.

DataCutter [2] is an application framework providing support for developing data-intensive applications that make use of scientific datasets in remote/archival storage systems across a wide-area network. DataCutter uses distributed processes to carry out a rich set of queries and application specific data transformations. DataCutter also provides support for sub-setting very large datasets through multi-dimensional range queries. The programming model of DataCutter is the filter-stream one, where applications are represented by a collection of filters connected by streams, and each filter performs some discrete function. A filter can also process multiple logically distinct portions of the total workload. This is referred to as a unit-of-work, and provides an explicit time when adaptation decisions may be made while an application is running.

The Knowledge Grid [5] is a software infrastructure for Parallel and Distributed Knowledge Discovery (PDKD). The Knowledge Grid uses basic Grid services such as communication, authentication, information, and resource management to build more specific PDKD tools and services. Knowledge Grid services are organized into two layers: core K-Grid layer, which is built on top of generic Grid services, and high level K-Grid layer, which is implemented over the core layer. The core K-Grid layer comprises services for (*i*) managing metadata describing data sources, data mining software, results of computations, data and results manipulation tools, execution plans, etc.; (*ii*) finding mappings between execution plans and available resources on the Grid, satisfying application requirements. The high-level K-Grid layer comprises instead services for building and executing PDKD computations. Such services are targeted to (*i*) searching, selection, extraction, transformation and delivery of data to be mined and data mining tools and algorithms; (*ii*) generation of possible execution plans on the basis of application requirements; (*iii*) generation and visualization of PDKD results.

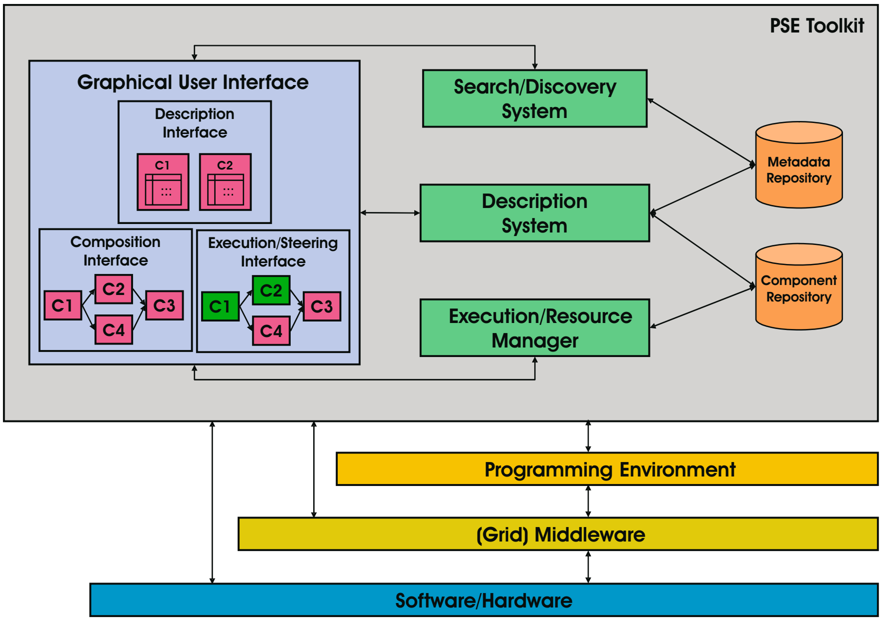


Fig. 1. A reference architecture of a PSE Toolkit

3 A Reference Architecture for a Grid-Based PSE Toolkit

As mentioned in the previous section, PSE portability, extensibility, and flexibility can be provided through the use of a PSE Toolkit. In this section, we identify the main components of a PSE Toolkit and describe how these components should interact to implement PSEs in a distributed setting.

A minimal reference architecture of a PSE Toolkit should include:

- A *Graphical User Interface*;
- A *Repository* of usable components, including user-defined applications;
- A metadata-based *Description System*, possibly based on application-domain ontologies;
- A *Metadata Repository*, possibly including an ontology repository;
- A *Search/Discovery System*;
- An *Execution/Resource Manager*.

Figure 1 shows all these components and the interactions among them to compose the architecture. In particular, the Component Repository represents a library of objects used to build PSE applications, the Metadata Repository is a knowledge base describing such library, whereas the remaining components are services used to search, design, and execute PSE applications. Details of each component are now discussed.

Graphical User Interface. It allows for:

- The description of available components, the semi-automatic construction of the associated metadata and their publishing. To this aim, the GUI interacts with the Description System. These actions can be driven by the underlying ontology that helps user in classifying components.
- The construction of applications and their debugging. For doing this, it goes through the following steps:
  1. Interaction with the Search and Discovery System for letting users pose queries and collect results about available components potentially usable for the PSE composition, or directly browse the metadata repository. As before, both querying and browsing can be guided by ontology.
  2. Design of an application through visual facilities.
  3. Interactive validation and debugging of the designed application.
  4. Invocation of the Execution Manager.
- The execution, dynamic steering of applications and the visualization of results. To this end, the GUI interacts with the Execution Manager to monitor the execution and to show the results, and it gives the user the possibility to steer the application.
- The storing of useful applications and results, annotated with domain knowledge. In this way, applications and their associated domain knowledge can be shared among PSE users.

In order to seamlessly integrate the facilities of external tools, the Graphical User Interface must also allow for the direct use of their native GUIs. This capability is fundamental, e.g., when external tools are provided through their native GUIs only, or when their integration in the Toolkit's GUI would result in excessive complexity.

Component Repository. The repository of usable components holds the basic elements/modules that are used to build a PSE. Examples of such components are software tools, data files, archives, remote data sources, libraries, etc. The repository must be able to integrate a possibly pre-existing one provided by the programming environment used to implement the PSE Toolkit, also comprising components defined by different languages and/or tools. The Component Repository must be able to manage dynamic connection/disconnection of Grid resources offering components. In fact, in this case computing nodes and network connections must be considered as components that can be used in the implementation of a PSE. The component repository can include complete user-defined applications that, when published in the repository, enhance the PSE capability to solve specific problems.

Metadata Repository. The Metadata Repository stores information about components in the Component Repository. Such information comprises component owners and providers, access modalities, interfaces, performances (offered or required), usage history, availability and cost. For the description of component interfaces, the PSE Toolkit provides an interface definition language (*IDL*) with which every component must comply, in order to be usable by the Toolkit. Thus, a component provider either makes it directly accessible with the IDL, or describes in the associated metadata how to translate the invocations written using the Toolkit IDL into actions to be taken and/or into invocations written in the native language of the component. Finally, also reusable *solutions*, i.e. modules of applications for solving particular sub-problems, that are typical for PSEs, can be managed and described as resources. The Metadata Repository can be implemented by using a Domain Ontology, i.e. a conceptualization, in a standard format, of component metadata, utilization, and relationships. Such ontology can be extended to describe user-defined applications.

Description System. The Description System must be able to offer a clear description of each element a PSE can be composed of; in the proposed architecture, it is based on the Metadata Repository. An important issue to be considered for the Description System is the use of ontologies for enriching the semantics of component descriptions. This layer extends the metadata facilities and can provide the user with a semantic-oriented view of resources. Basic services of the Description System are *component classification* through taxonomies, *component annotations*, for example indicating which problem they are useful for, and *structured metadata description*, for example by using standard, searchable data. Whenever new components/applications are added to the Component Repository, new knowledge is added to the Metadata Repository through the Description System.

Search/Discovery System. The Search and Discovery System accesses the Metadata Repository to search and find all the resources in the environment where the Toolkit runs, which are available to a user for composing a PSE. While the implementation of this system on a sequential machine is straightforward, in a parallel/distributed setting the Search/Discovery System should be designed as a distributed service able to search resources on a fixed set of machines. Moreover, in a Grid computing environment such a system should be able to search over a dynamically changing heterogeneous collection of computers. Basic services of the Search/Discovery System are: *ontology-based search* of components, that allows for searching components on the basis of belonging taxonomies as well as specifying constraints on their functions, and the *key-based search*. The former operates on a portion of the knowledge base selected through the ontology, whereas the latter operates on the entire knowledge base.

Execution/Resource Manager. The PSE Toolkit Execution/Resource Manager is the run-time support of generated PSEs. Its complexity depends on the components involved in the PSE composition and in the hardware/software architecture used. The Execution/Resource Manager must tackle several issues, e.g. the selection of the actual component instances to be used, the suitable assignment of processes to heterogeneous computing resources (scheduling), and the distributed coordination of their execution, possibly adapting the running applications to run-time (unpredictable) changes in the underlying hardware/software infrastructure. If a Grid computing architecture is considered, the Execution/Resource Manager, besides interaction with the software/hardware system, has a tight connection with the Grid fabric environment and with the Grid middleware. Note that the Execution/Resource Manager is also responsible for performing all the actions needed to activate the available components, driven by their metadata. These actions can comprise, e.g., compiling a piece of code or a library before using them, or launching a program onto a particular virtual machine. The execution manager must have tight relationships with the run-time support of the programming languages used for coding the components.

4 Conclusions

We have proposed a reference architecture for a Grid-based PSE Toolkit, whose main characteristics are: (*i*) the existence of a knowledge base built around pre-existing components (software and data sources); (*ii*) the composition of the application, conducted through the searching and browsing of the knowledge base; and (*iii*) the distributed and coordinated execution of the application on the selected distributed platform, that can be a traditional distributed system or a Grid. A more complete description of the architecture and of the related topics can be found in [4]. We are working to develop a prototype version of the PSE toolkit for a Grid environment based on Globus Toolkit 3. This will allow us also to investigate how to exploit Grid services for implementing the PSE Toolkit functionality.

References

1. E. Akarsu, G. Fox, W. Furmanski, T. Haupt. Webflow - High-Level Programming Environment and Visual Authoring Toolkit for High Performance Distributed Computing. *Supercomputing'98*, Florida, November 1998.
2. M. D. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz. DataCutter: Middleware for Filtering Very Large Scientific Datasets on Archival Storage Systems. *MASS2000*, pages 119-133. National Aeronautics and Space Administration, Mar. 2000. NASA/CP 2000-209888.
3. The cactus code website. *http://www.cactuscode.org*.
4. M.Cannataro, C. Comito, A. Congiusta, G.Folino, C. Mastroianni, A.Pugliese, G. Spezzano, D. Talia, P. Veltri. Grid-based PSE Toolkits for Multidisciplinary Applications. RT-ICAR-CS-03-10. ICAR-CNR Technical Report, October 2003.
5. M. Cannataro, D. Talia. KNOWLEDGE GRID: An Architecture for Distributed Knowledge Discovery. *Communications of the ACM*, January 2003.
6. J. C. Cunha. Future Trends in Distributed Applications and PSEs. Talk held at the *Euresco Conference on Advanced Environments and Tools for High Performance Computing*, Albufeira (Portugal), 2003.
7. E. Gallopoulos, E. N. Houstis, J. Rice. Computer as Thinker/Doer: Problem-Solving Environments for Computational Science. *IEEE Computational Science and Engineering*, vol.1, n. 2, 1994.
8. The Globus Project. *http://www.globus.org*.
9. The Grid Portal Toolkit. *http://gridport.npaci.edu*.
10. S. Krishnan, R. Bramley, D. Gannon, M. Govindaraju, R. Indurkar, A. Slominski, B. Temko, R. Alkire, T. Drews, E. Webb, and J. Alameda. The XCAT Science Portal. High-Performance Computing and Networking Conference (*SC*), 2001.
11. K. Schuchardt, B. Didier, G. Black. Ecce - a Problem-Solving Environment's Evolution toward Grid Services and a Web Architecture. *Concurrency and computation: practice and experience*, vol. 14, pages 1221-1239, 2002.
12. D. Walker, O. F. Rana, M. Li, M. S. Shields, Y. Huang. The Software Architecture of a Distributed Problem-Solving Environment. *Concurrency: Practice and Experience*, vol. 12, No. 15, pages 1455-1480, December 2000.

An Expert Assistant for Computer Aided Parallelization

Gabriele Jost[1,⋆], Robert Chun[2], Haoqiang Jin[1], Jesus Labarta[3], and Judit Gimenez[3]

[1] NAS Division, NASA Ames Research Center
Moffett Field, CA 94035-1000, USA
{gjost,hjin}@nas.nasa.gov

[2] Computer Science Department, San Jose State University
San Jose, CA 95192, USA
Robert.Chun@sjsu.edu

[3] European Center for Parallelism in Barcelona
Technical University of Catalonia (CEPBA-UPC),
cr. Jordi Girona 1-3, Modul D6, 08034 – Barcelona, Spain
{jesus,judit}@cepba.upc.es

Abstract. The prototype implementation of an expert system was developed to assist the user in the computer aided parallelization process. The system interfaces to tools for automatic parallelization and performance analysis. By fusing static program structure information and dynamic performance analysis data the expert system can help the user to filter, correlate, and interpret the data gathered by the existing tools. Sections of the code that show poor performance and require further attention are rapidly identified and suggestions for improvements are presented to the user. In this paper we describe the components of the expert system and discuss its interface to the existing tools. We present a case study to demonstrate the successful use in full scale scientific applications.

1 Introduction

When porting an application to a parallel computer architecture, the program developer usually goes through several cycles of code transformations followed by performance analysis to check the efficiency of parallelization. A variety of software tools have been developed to aid the programmer in this challenging task. A parallelization tool will usually provide static source code analysis information to determine if and how parallelization is possible. A performance analysis tool provides dynamic runtime information in order to determine the efficiency of time consuming code fragments. The programmer must take an active role in driving the analysis tools and in interpreting and correlating their results to make the proper code transformations to improve parallelism. For large scientific applications, the static and dynamic analysis results are typically very complex and their examination can pose a daunting task for the user. In this paper we describe the coupling of two mature analysis tools by an expert system. The tools under consideration

⋆ The author is an employee of Computer Sciences Corporation.

are the CAPO [3] parallelization tool and the Paraver [12] performance analysis system. The techniques described in this paper are applicable to many parallel programming paradigms, but we will restrict our discussion to OpenMP [10] parallelization for shared memory computer architectures.

OpenMP supports loop level parallelization in the form of compiler directives. The program developer has to insert the directives and specify the scope of the variables. The CAPO parallelization tool was developed to aid the programmer in this task. CAPO automates OpenMP directive insertion into existing Fortran codes and allows user interaction for an efficient placement of the directives. This is achieved by use of extensive interprocedural analysis from CAPTools [2] (now known as ParaWise) which was developed at the University of Greenwich. Dependence analysis results and other static program structure information are stored in an application database. CAPO provides an extensive set of browsers which allow the user to examine the information and to provide additional information, such as values of runtime parameters. While this enables a very precise analysis and the generation of highly efficient parallel code, it also introduces challenges for the program developer: A full scale scientific application will usually contain many subroutines, loops, and variables, yielding a large amount of analysis. Since it is impossible to examine all dependences, the programmer has to make a choice where to focus his efforts on.

Runtime performance analysis information provides means to help the user focus his optimization efforts. The Paraver performance analysis system is being developed and maintained at the European Center for Parallelism in Barcelona (CEPBA). Its major components are the tracing package OMPItrace [9], a graphical user interface to examine the traces based on time line views, and an analysis module for the computation of performance statistics. Profiling information can be displayed in form of tables or histograms. The statistics can be correlated and mapped onto other objects, such as subroutines and threads of execution. Just like in the case of the static application database, the performance trace of a large application will contain a plethora of information and the user is faced with the challenge of its meaningful interpretation.

An expert analyst will usually go through repetitive inspections of application data dependences and performance statistics. The novice user will often not even know where to start and what to look for. In our approach we try to address the needs of expert as well as novice user. The rest of the paper is structured as follows: In Section 2 we describe the prototype implementation of our expert system. In Section 3 we present a case study to show the usefulness of the system. In Section 4 we discuss some related work and draw our conclusions in Section 5.

2 The Expert System Prototype Implementation

An intelligent computerized parallelization advisor was developed using an expert system implemented in CLIPS ("C" Language Integrated Production System) [1]. By performing data fusion on the static and dynamic analysis, the expert system can help the user to filter, correlate, and interpret the data. Relevant performance indices are automatically calculated and correlated with program structure information. The overall architecture of our program environment is depicted in Figure 1. The application database contain-

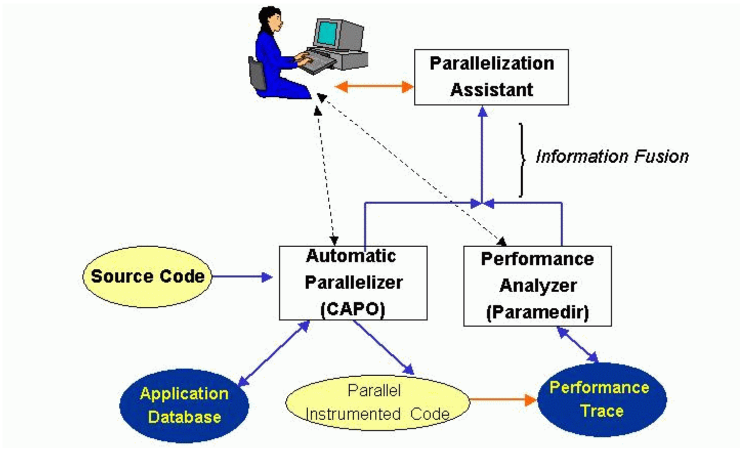

Parallelization Assistant
Information Fusion
Source Code
Automatic Parallelizer (CAPO)
Performance Analyzer (Paramedir)
Application Database
Parallel Instrumented Code
Performance Trace

Fig. 1. Architecture of the expert system programming environment. The Parallelization Assistant Expert System fuses program structure knowledge and performance trace information to aid the user in narrowing down performance problems. The user can still interact directly with the parallelization and performance analysis tools for fine tuning of the application's performance

ing static program structure information such as dependence analysis results is gathered during a CAPO session. The information stored in the application database is used to generate parallelized source code. In the following we describe the steps performed by the expert system in order to evaluate the efficiency of the generated code.

Instrumenting the Parallel Source Code for Performance Analysis: The expert system prototype uses the Paraver OMPItrace package for instrumentation and tracing of the application. OMPItrace generates traces containing time stamped events during the program's execution. Tracing of some events occurs automatically and does not require any modification of the source code or relinking of the application with special libraries. Examples of such events are

– entry end exit of compiler generated routines for the execution of parallelized code segments,
– entry and exit of parallelization runtime libraries, such as the OpenMP runtime library,
– hardware counters, if they are available.

User level subroutines have to be manually instrumented in the source code. Instrumentation of all subroutines may introduce a lot of overhead. For critical code segments, on the other hand, it is often desirable to obtain information at a finer granularity than a subroutine call. The prototype expert system uses CAPO's source code transformation capabilities and the program structure information from the application database to selectively instrument the code. Only routines that are not contained within a parallel region or a parallel loop and that contain at least one DO-loop chosen for instrumentation. In addition to this outermost serial loops are also instrumented. More details on the automatic selective instrumentation can be found in [4].

Automatic Retrieval of Performance Indices: We have extended the Paraver system by Paramedir, a non-graphical command line interface to the analysis module. The specification of Paraver trace file views and metric calculations can be saved to reusable configuration files. Paramedir accepts the same trace and configuration files as Paraver. In this way the same information can be captured in both systems. Paramedir is invoked in batch mode, providing a trace file and a configuration file which specifies the metric calculation as input. The generated output is an ASCII file containing the calculated statistics. This supports the programmability of performance analysis in the sense that complex performance metrics, determined by an expert user, can be automatically computed and processed. The detailed human driven analysis can thus be translated into rules suitable for processing by an expert system. Details on Paramedir can be found in [5]. Examples of metrics of the instrumented code segments which are automatically calculated by the current prototype are:

– **timing profiles:** the percentage of time which is spent in subroutines and loops,
– **sequential sections:** time that the master thread spends outside of parallel regions.
– **useful time:** time that the applications spends running user code and not idling, synchronizing, or other parallelization introduced overhead.
– **workload balance:** the coefficient of variation over the threads in the number of instructions within useful time,
– **parallelization granularity:** the average duration of a parallel work sharing chunk,
– **estimated parallel efficiency:** the speed-up divided by the number of threads. The execution time for a run on one thread is estimated using the useful time of all threads of the parallel run.

These metrics are automatically calculated using Paramedir and are stored in a table. They are facts for the rule based analysis of the expert system.

Information Fusion: After calculating the performance indices, the expert system extracts program structure information from the application database. At this point the main interest is the optimal placement of OpenMP directives. The prototype implementation retrieves the type of the instrumented code segment, such as loop or subroutine, the loop identifier, and the type of the loop, such as being parallel or sequential.

Rule Based Analysis: The expert system uses observations to infer reasons for poor performance. The observations are a list of facts about the calculated performance metrics and static program analysis for the calculated code segments, such as described in the previous paragraph. Examples are: “the subroutine takes 50% of the execution time” or “the variation of executed instructions among the threads is high”. The metrics are compared to empirically determined threshold values in order to determine what is high and what is low. Conclusions are inferred through a set of rules that interpret certain patterns as reasons for poor performance. An example would be a rule of the form:

– **If:** The code segment takes a large amount of time **and** the parallel efficiency is low **and** there are large sequential sections **and** the granularity of the parallelization is fine
– **Then:** Try to move the parallelization to an outer level

An illustration of the reasoning is given in Figure 2. The conclusions are presented to the user either as ASCII text or via the CAPO user interface. This will be discussed in the case study presented in Section 3.The rules are stored in a knowledge base. The

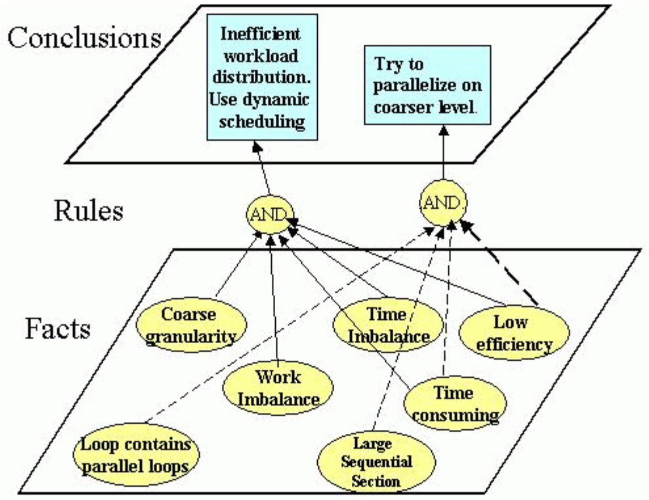


Fig. 2. Example of facts and conclusions within the expert system. A set of rules is applied to the observed facts to determine patterns of causes for poor performance

expert system employs a data driven, forward chaining approach where a conclusion is infered given all asserted facts.

3 A Case Study

The parallelization assistant expert system was tested on several full scale scientific applications. In order to give a flavor of its usability we present the analysis of the PSAS Conjugate Gradient Solver. The code is a component of the Goddard EOS (Earth Observing Systems) Data Assimilation System. It contains a nested conjugate gradient solver implemented in Fortran 90. The code showed little speed-up on an SGI Origin 3000 when increasing the number of threads from 1 to 4. A sample trace was collected on an SGI Origin 3000 for a run on 4 threads. Figure 3 shows three Paraver timeline views which were used for the calculation of performance metrics. The top image shows the time that the application spends in different subroutines. The different shadings indicate time spent in different routines. It turned out that there were two major time consuming routines:

- the conjugate gradient solver, indicated by the lighter shading in the timeline,
- the symmetric covariance matrix vector multiply, indicated by the darker shading in the timeline

The middle image in Figure 3 shows the useful computation time spent within different parallel loops. This is a thread level view, where the different shading indicate different parallel loops. No shading indicates time outside of computations. Visual inspection immediately shows an imbalance between the threads, with thread number 1 spending a lot more time inside parallel computations. The reason for the imbalance can be inferred from the view displayed in the bottom image in Figure 3. It shows a time line view

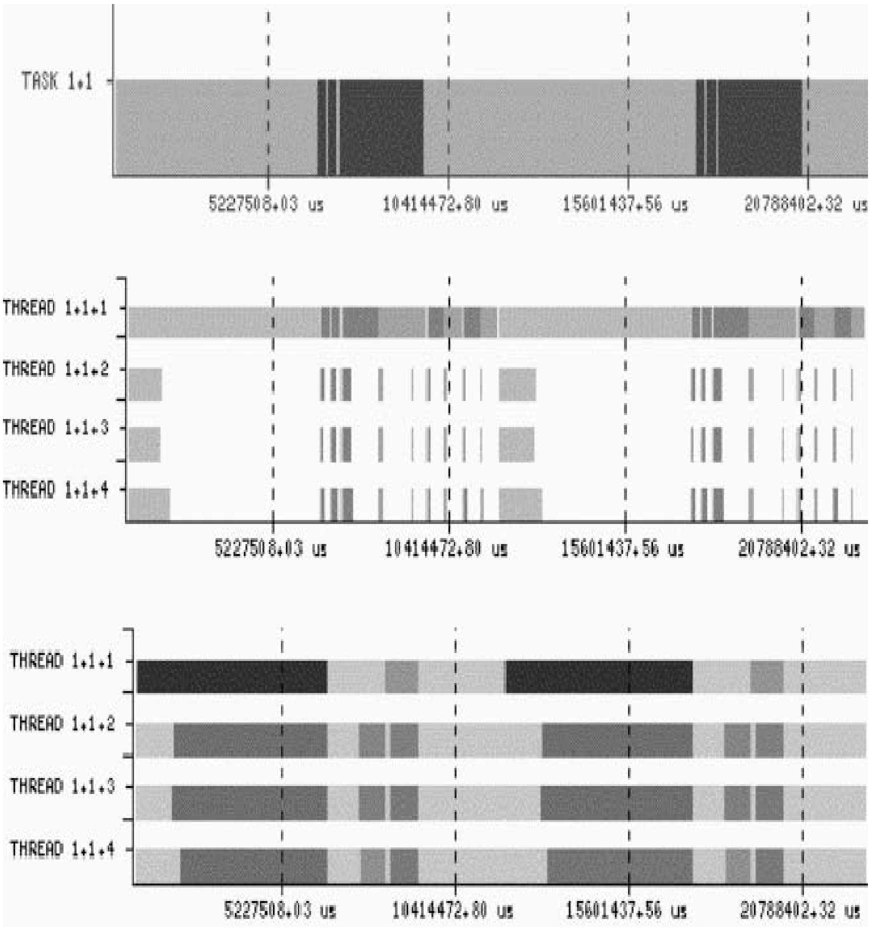
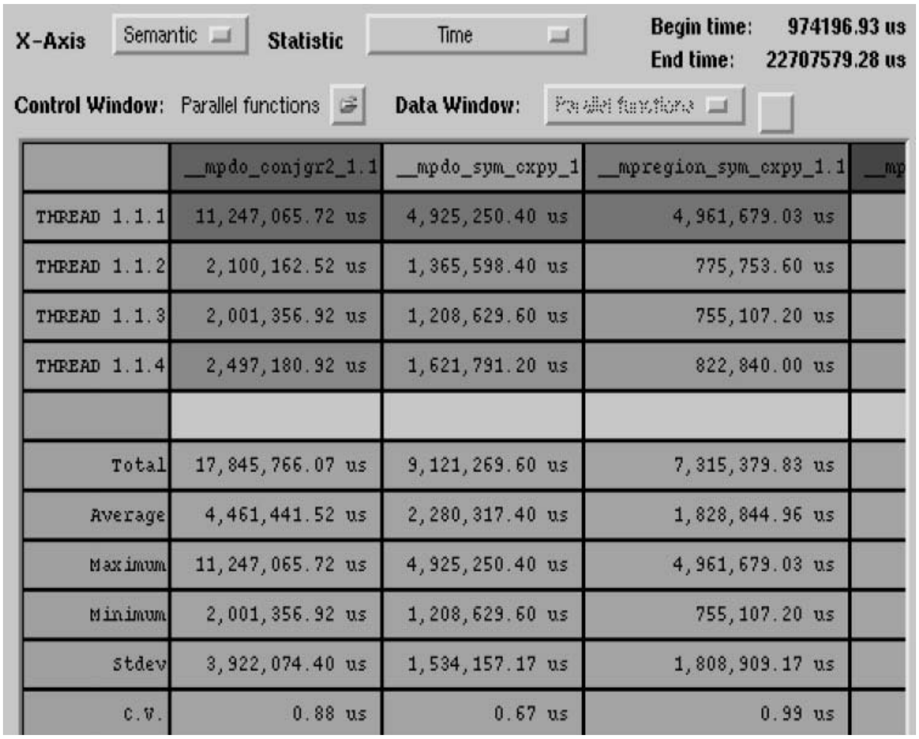


Fig. 3. Three Paraver timeline views for a run on 4 threads: The top image shows the time spent in different subroutines on application level. The horizontal axis represents the time. Different shadings indicate time in different subroutines. The middle image shows time spent in useful computations within parallel loops on thread level, displaying a time line for each of the 4 threads. Useful computation time is shaded, non-useful time is white. The view shows an obvious imbalance, with the master thread spending more time in parallel computations. The bottom image shows the number of executed instructions for each thread. Darker shading indicates a higher number of executed instructions. This view indicates an imbalance in the computational workload among the threads

on thread level of the number of instructions between traced events. A darker shading indicates a higher value of the number of instructions. Comparison of the time line views clearly shows that thread 1 executes many more instructions during the useful time within parallel loops. A second important observation is, that a relatively large number of instructions are executed by threads 2,3 and 4 during non-useful time. These



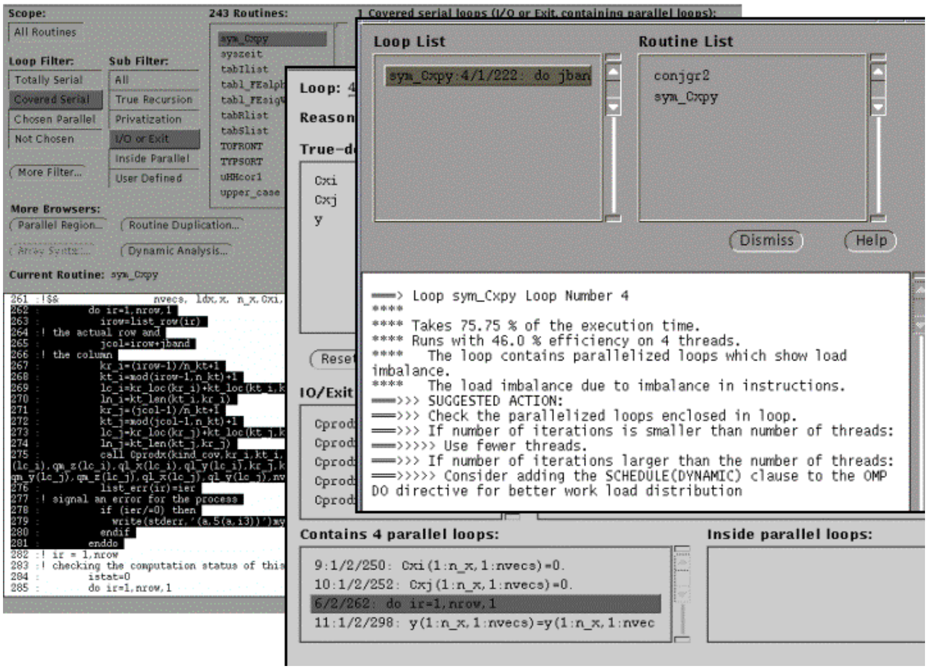
Begin time: 974196.93 us
End time: 22707579.28 us

	__mpdo_conjgr2_1.1	__mpdo_sym_cxpy_1	__mpregion_sym_cxpy_1.1	__mp
THREAD 1.1.1	11,247,065.72 us	4,925,250.40 us	4,961,679.03 us	
THREAD 1.1.2	2,100,162.52 us	1,365,598.40 us	775,753.60 us	
THREAD 1.1.3	2,001,356.92 us	1,208,629.60 us	755,107.20 us	
THREAD 1.1.4	2,497,180.92 us	1,621,791.20 us	822,840.00 us	
Total	17,845,766.07 us	9,121,269.60 us	7,315,379.83 us	
Average	4,461,441.52 us	2,280,317.40 us	1,828,844.96 us	
Maximum	11,247,065.72 us	4,925,250.40 us	4,961,679.03 us	
Minimum	2,001,356.92 us	1,208,629.60 us	755,107.20 us	
Stdev	3,922,074.40 us	1,534,157.17 us	1,808,909.17 us	
C.V.	0.88 us	0.67 us	0.99 us	

Fig. 4. Paraver Analysis showing the number of instructions during useful computations. The hardware counter value is mapped onto threads and parallel regions. The analysis shows a large variation across the threads, indicating an unbalance of the computational workload

are typically instructions executed by threads when idling while waiting for work. It is important to exclude these instructions when comparing the computational workload among the threads. A Paraver analysis of the instructions only during useful time is displayed in Figure 4, where the instruction counter value is correlated to the thread numbers and the parallel loops, indicating a large variation in useful instructions among the threads.

The reason for the imbalance is that the amount of calculations per iteration differs greatly. By default, the iterations of a parallel loop are distributed block-wise among the threads, which is a good strategy if the amount of work per iteration is approximately the same. In the current case, thread number 1 ended up with all of the computationally intensive iterations, leaving the other threads idle for a large amount of time. The OpenMP directives provide clauses to schedule work dynamically, so that whenever a thread finished its chunk of work, it is assigned the next chunk. This strategy should only be used if the granularity of the work chunks is sufficiently coarse. After calculating the performance metrics described in Section 2 the expert system was able to detect the imbalance and its possible cause. The rule which fired for this particular case was:
]



```
:!$&                    nvecs, ldx,x, n_x,Cxi,
:          do ir=1,nrow,1
:             irow=list_row(ir)
:! the actual row and
:             jcol=irow+jband
:! the column
:             kr_i=(irow-1)/n_kt+1
:             kt_i=mod(irow-1,n_kt)+1
:             lc_i=kr_loc(kr_i)+kt_loc(kt_i,k
:             ln_i=kt_len(kt_i,kr_i)
:             kr_j=(jcol-1)/n_kt+1
:             kt_j=mod(jcol-1,n_kt)+1
:             lc_j=kr_loc(kr_j)+kt_loc(kt_j,k
:             ln_j=kt_len(kt_j,kr_j)
:             call Cprodx(kind_cov,kr_i,kt_i,
(lc_i),qm_z(lc_i),ql_x(lc_i),ql_y(lc_i),kr_j,k
qm_y(lc_j),qm_z(lc_j),ql_x(lc_j),ql_y(lc_j),nv
:             list_err(ir)=ier
:! signal an error for the process
:             if (ier/=0) then
:                write(stderr,'(a,5(a,i3))')my
:             endif
:          enddo
:! ir = 1,nrow
:! checking the computation status of this
:          istat=0
:          do ir=1,nrow,1
```

```
===> Loop sym_Cxpy Loop Number 4
****
**** Takes 75.75 % of the execution time.
**** Runs with 46.0 % efficiency on 4 threads.
****   The loop contains parallelized loops which show load
imbalance.
****   The load imbalance due to imbalance in instructions.
===>>> SUGGESTED ACTION:
===>>> Check the parallelized loops enclosed in loop.
===>>> If number of iterations is smaller than number of threads:
===>>>>> Use fewer threads.
===>>> If number of iterations larger than the number of threads:
===>>>>> Consider adding the SCHEDULE(DYNAMIC) clause to the OMP
DO directive for better work load distribution
```

Fig. 5. The expert system analysis can be displayed in CAPO's Dynamic Analysis Window. Selecting a particular routine or loop, the corresponding source code is highlighted in the Directives Browser window. The Why Window displays information about the status of the parallelization of the code, such as the scope of variables or dependences that prevent parallelization

– **If:** The code segment takes a large amount of time **and** the parallel efficiency is low **and** there is a large variation in the amount of useful computation time **and** there is a large variation in the number of instructions within the parallelized loops **and** the granularity of the parallelization is sufficiently large
– **Then:** Try dynamic scheduling in order to achieve a better work load balance.

The expert system analysis output is generated as an ASCII file. The user has the possibility to display the analysis results using the CAPO directives browser. This integrates the dynamic performance analysis data with program structure information. An example is shown in Figure 5. The user is presented with a list of time consuming loops and subroutines. Selecting an item from the list displays the corresponding expert system analysis and highlights the corresponding source in the directive browser window. For the current case it helps the user to identify loops which suffer from workload imbalance. The expert system analysis provides guidance through the extensive set of CAPO browsers.

4 Related Work

There are several research efforts on the way with the goal to automate performance analysis to integrate performance analysis and parallelization. We can only name a few of these projects. KOJAK [6] is a collaborative project of the University of Tennessee and the Research Centre Juelich for the development of a generic automatic performance analysis environment for parallel programs aiming at the automatic detection of performance bottlenecks. The Paradyn Performance Consultant [8] automatically searches for a set of performance bottlenecks. The system dynamically instruments the application in order to collect performance traces. The URSA MINOR project [11] at Purdue University uses program analysis information as well as performance trace data in order to guide the user through the program optimization process. The SUIF Explorer [7,13] Parallelization Guru developed at Stanford University uses profiling data to bring the user's attention to the most time consuming sections of the code. Our approach using an expert system differs from the previous work in that we are integrating two mature tools. Our rule based approach takes advantage of the high degree of flexibility in collecting and analyzing information provided by the underlying tools.

5 Conclusions

We have built the prototype of an expert assistant to integrate two mature tools for computer aided parallelization and performance analysis. The system helps the user in navigating and interpreting the static program structure information, as well as the accompanying dynamic run-time performance information so that more efficient optimization decisions can be made. It enables the users to focus their tuning efforts on the sections of code that will yield the largest performance gains with the least amount of recoding.

Using the expert system approach in a case study has demonstrated how to fuse data from the static and dynamic analysis to provide automated correlation and filtering of this information before conveying it to the user. The first conclusion we draw is that a number of relatively simple rules can capture the essence of the human expert's heuristics needed to narrow down a parallelization related performance problem. Secondly, we found it to be very important to be able to switch to the direct usage of the tools at any point during the analysis process. The expert system analysis rapidly guides the user to code segments, views, and effects that require further detailed analysis with either CAPO or Paraver. This detailed analysis will in turn often lead to the design of new rules which can then be included in the automated process.

Acknowledgments

This work was supported by NASA contract DTTS59-99-D-00437/A61812D with Computer Sciences Corporation/AMTI, by the NASA Faculty Fellowship Program, and by the Spanish Ministry of Science and Technology, by the European Union FEDER program under contract TIC2001-0995-C02-01, and by the European Center for Parallelism of Barcelona (CEPBA). We thank the CAPTools development team for many helpful discussions and the continued support of our work.

References

1. CLIPS: A Tool for Building Expert Systems, http://www.ghg.net/clips/CLIPS.html.
2. C.S. Ierotheou, S.P. Johnson, M. Cross and P. Leggett, "Computer Aided Parallelisation Tools (CAPTools), Conceptual Overview and Performance on the Parallelisation of Structured Mesh Codes," Parallel Computing, 22 (1996) 163-195.
http://www.parallelsp.com/parawise.htm.
3. H. Jin, M. Frumkin and J. Yan, "Automatic Generation of OpenMP Directives and Its Application to Computational Fluid Dynamics Codes," Proceedings of Third International Symposium on High Performance Computing (ISHPC2000), Tokyo, Japan, October 16-18, 2000.
4. G. Jost, H. Jin, J. Labarta and J. Gimenez, "Interfacing Computer Aided Parallelization and Performance Analysis," Proceedings of the International Conference of Computational Science - ICCS03, Melbourne, Australia, June 2003.
5. G. Jost, J. Labarta and J. Gimenez, "Paramedir: A tool for programmable Performance Analysis", Proceedings of the International Conference of Computational Science - ICCS04, Krakow, Poland, June 2004.
6. Kit for Objective Judgment and Knowledge based Detection of Performance Bottlenecks, http://www.fz-juelich.de/zam/kojak/.
7. S. Liao, A. Diwan, R.P. Bosch, A. Ghuloum and M. Lam, "SUIF Explorer: An interactive and Interprocedural Parallelizer," 7th ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming, Atlanta, Georgia, (1999), 37-48.
8. B.P. Miller, M.D. Callaghan, J. M. Cargille, J. K. Hollingsworth, R. B. Irvin, K.L. Karavanic, K. Kunchithhapdam and T. Newhall, "The Paradyn Parallel Performance Measurement Tools," IEEE Computer 28, 11, pp.37-47 (1995).
9. OMPItrace User's Guide, http://www.cepba.upc.es/paraver/manual_i.htm.
10. OpenMP Fortran/C Application Program Interface, http://www.openmp.org/.
11. I. Park, M. J. Voss, B. Armstrong and R. Eigenmann, "Supporting Users' Reasoning in Performance Evaluation and Tuning of Parallel Applications," Proceedings of PDCS'2000, Las Vegas, NV, 2000.
12. Paraver, http://www.cepba.upc.es/paraver/.
13. SUIF Compiler System. http://suif.stanford.edu/.

Scalable Middleware Environment for Agent-Based Internet Applications

Benno J. Overeinder[*] and Frances M.T. Brazier

Department of Computer Science, Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{bjo,frances}@cs.vu.nl

Abstract. The AgentScape middleware is designed to support deployment of agent-based applications on Internet-scale distributed systems. With the design of AgentScape, three dimensions of scalability are considered: size of distributed system, geographical distance between resources, and number of administrative domains. This paper reports on the AgentScape design requirements and decisions, its architecture, and its components.

1 Introduction

Agent-based Internet applications are by design autonomous, self-planning, and often self-coordinating distributed applications. They rely on the availability of aggregated resources and services on the Internet to perform complex tasks. However, current technology restricts developers options for large-scale deployment: there are problems with the heterogeneity of both computational resources and services, fault tolerance, management of the distributed applications, and geographical distance with implied latency and bandwidth limitation.

To facilitate agent-based applications, a middleware infrastructure is needed to support mobility, security, fault tolerance, distributed resource and service management, and interaction with services. Such middleware makes it possible for agents to perform their tasks, to communicate, to migrate, etc.; but also implements security mechanisms to, for example, sandbox agents to prevent malicious code harm the local machine, or vice versa, protect an agent from tampering by a malicious host.

The AgentScape middleware infrastructure is designed to this purpose. Its multi-layered design provides minimal but sufficient support at each level. The paper presents the AgentScape design and implementation, with a discussion on decisions made, and concludes with future directions.

2 Scalable Multi-layered Design of Agent Middleware

A number of high-level agent concepts are first introduced to define their role in the paper. The requirements for scalable agent middleware are then discussed, followed by the current software architecture of the AgentScape middleware [6].

[*] This research is supported by the NLnet Foundation, http://www.nlnet.nl/

2.1 Concepts

Within AgentScape, *locations* exist, *agents* are active entities, and *services* are external software systems accessed by agents hosted by AgentScape middleware. Agents in AgentScape are defined according to the weak notion of agency [7]: (*i*) autonomy: agents control their own processes; (*ii*) social ability: ability to communicate and cooperate; (*iii*) reactiveness: ability to receive information and respond; (*iv*) pro-activeness: ability to take the initiative.

Agents may communicate with other agents and may access services. Agents may migrate from one location to another location. Agents may create and delete agents; agents can start and stop service access. All operations of agents are modulo authorization and security precautions, e.g., an agent is allowed to start a service if it has the appropriate credentials (ownership, authorization, access to resources, etc.).

Mechanisms are provided to make external services available to agents hosted by AgentScape middleware. For example, external services are wrapped and presented as Web services, using SOAP/WSDL generated dynamic interfaces.

A location is a "place" in which agents and services can reside. (see also Fig. 1). More precisely stated, agents and services are supported by agent servers and (Web) service gateways (respectively) which belong to a location. From the perspective of agents, agent servers give exclusive access to the AgentScape middleware. Similarly, service gateways provide access to services external to AgentScape.

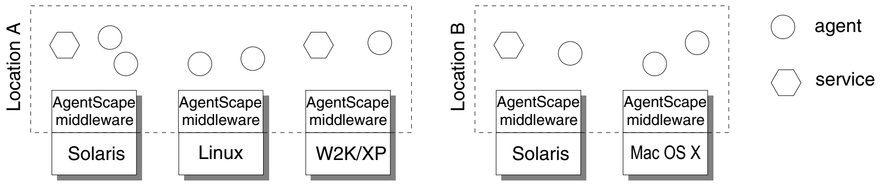


Fig. 1. Conceptual model of the AgentScape middleware environment

2.2 Requirements

Successful deployment of Internet applications has to take dimensions of scale into account [5]. Applications and systems on Internet-scale networks have to include mechanisms to deal with: (*i*) number of entities in a distributed system: agents (create, delete, migration), resources (add and delete resources to/from distributed platform, allocate resources), lookup service (resolve identifiers of entities to contact address, etc.); (*ii*) geographical size: the distance between different resources (communication latency and bandwidth); (*iii*) number of administrative domains: security mechanisms for authentication and authorization.

The design goals set in the AgentScape project are to provide efficient support for distributed systems that scale along the dimensions outlined above. A distinction is made between agent application level support and agent middleware mechanisms. At the agent application level, functionality to develop and implement scalable applications must be made available. The application programming interface must reflect this, e.g., to allow for latency hiding and mobility, or give access to scalable services in the middleware layer. The agent middleware's task is to implement the functionality provided by the application programming interface and the available services. This implies, amongst others, support for asynchronous communication mechanisms for latency hiding, include scalable services for name resolution (lookup service), and an effective management system that scales with the number of agents and resources.

2.3 Software Architecture

The leading principle in the design of the AgentScape middleware is to develop a minimal but sufficient open agent platform that can be extended to incorporate new functionality or adopt (new) standards into the platform. The multiple code base requirement, e.g., supporting agents and services developed in various languages, makes that language specific solutions or mechanisms cannot be used.

This design principle resulted in a layered agent middleware, with a small *middleware kernel* implementing basic mechanisms and high-level *middleware services* implementing agent platform specific functionality and policies (see Fig. 2). This approach simplifies the design of the AgentScape kernel and makes the kernel less vulnerable to errors or improper functioning. A minimal set of middleware services are agent servers, host managers, and location managers. The current, more extensive set includes a lookup service and a web service gateway.

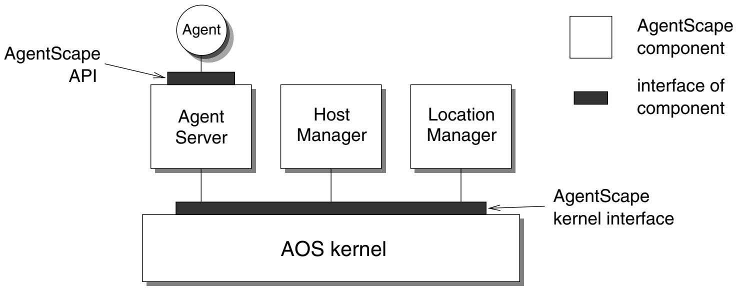


Fig. 2. The AgentScape software architecture

Minimal Set of Services
Middleware services can interact *only* with the local middleware kernel. That is, all interprocess communication between agent servers and web service gateways is exclusively via their local middleware kernel. The kernel either directly handles the transaction (local operation) or forwards the request messages to the destination AgentScape kernel (remote operation).

The agent server gives an agent access to the AgentScape middleware layer (see also Fig. 2). Multiple code base support in AgentScape is realized by providing different agent servers per code base. For security considerations, it is important to note that an agent is "sandboxed" by an agent server. For Java agents this is realized by the JVM, for Python (or other interpreted scripting languages like Safe-Tcl) by the interpreter, and for C or C++ (binary code) agents are "jailed".

A location manager is the coordinating entity in an AgentScape location (thus managing one or more hosts in one location). The host manager manages and coordinates activities on a host. The host manager acts as the local representative of the location manager, but is also responsible for local (at the host) resource access and management. The policies and mechanisms of the location and host manager infrastructure are based on negotiation and service level agreements [4].

Additional Services
Extensibility and interoperability with other systems are realized by additional middleware services. The web service gateway is a middleware service that provides controlled (by AgentScape middleware) access to SOAP/WSDL web services. Agents can obtain WSDL descriptions in various ways, e.g., via UDDI and can generate a stub from the WSDL document to access a web service. However, stub generation using the Axis WSDL2Java tool is slightly modified so that calls on the web service interface are directed to the AgentScape web service gateway. If access is authorized, the web service gateway performs the operation on the web service and routes the results transparently back to the agent that issued the call.

3 Implementation and Experiences

A number of implementation alternatives have been tested and evaluated, and development still evolves: the AgentScape project aims to provide both a scalable, robust agent platform, and a research vehicle to test and evaluate new ideas. The AgentScape middleware has been tested on different Linux distributions (Debian 3.1, Fedora Core 1, and Mandrake 9.2.1) and Microsoft Windows 2000 (service pack 4).

The current kernel is implemented in the Python programming language, allowing for rapid prototyping. All following middleware services available in the current AgentScape release have been implemented in Java: Java agent server, web service gateway, host manager, and a location manager. The current lookup service, implemented in Python, is a centralized service that basically does its job, but is not scalable and secure. A next generation lookup service is under development: a distributed peer-to-peer lookup service.

4 Related Work

Over the last few years, a number of agent platforms have been designed and implemented. Each design has its own set of design objectives and implementation decisions.

JADE is FIPA compliant agent platform [1]. In JADE, platform and containers are similar concepts as location and hosts in AgentScape. A JADE platform is a distributed agent platform (location), and with one or more containers (hosts). A special front end container listens for incoming messages from other platforms. The AgentScape location manager fulfills a similar function. The front end container is also involved in locating agents on other JADE platforms. In AgentScape this is a different (distributed) service. Mobility in JADE is limited to one platform (intra-platform), whereas in AgentScape, agents can migrate to any location.

Cougaar is a Java-based agent architecture that provides a survivable base on on which to deploy large-scale, robust distributed multi-agent systems [2]. The design goals are scalability, robustness, security, and modularity. Cougaar is not standards' compliant, and messages are encoded using Java object serialization. The lack of standards' compliance was in part due to Cougaar's complex planning language, which does not easily fit into the ACL format, and Cougaar's research focus of a highly scalable and robust system as opposed to interoperability.

DIET is an agent platform that addresses present limitations in terms of adaptability and scalability [3]. It provides an environment for an open, robust, adaptive and scalable agent ecosystem. The minimalistic "less is more" design approach of DIET is similar to AgentScape. The platform is implemented in Java. Mobility is implemented by state transfer only, so no code is migrated. Agents can only migrate if their classes are already in the local classpath of the target machine. Security is not specifically addressed in DIET.

5 Summary and Future Directions

The design rationale for scalability in AgentScape lies in the three dimensions as defined in Section 2.2. The integrated approach to solve the scalability problem is a unique signature of the AgentScape middleware. The small but extensible core of AgentScape allows for interoperability with other open systems.

Future directions in the AgentScape project are a new middleware kernel developed in Java and completion of the implementation of security model. Agent servers for binary agents (C and C++) and Python agents are being considered. A new decentralized lookup service based on distributed hash tables is also being developed.

References

1. F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with a FIPA-compliant agent framework. *Software: Practice and Experience*, 31(2):103–128, 2001.
2. A. Helsinger, M. Thome, and T. Wright. Cougaar: A scalable, distributed multi-agent architecture. In *Proceedings of the International Conference on Systems, Man and Cybernetics (IEEE SMC 2004)*, The Hague, The Netherlands, October 2004.
3. C. Hoile, F. Wang, E. Bonsma, and P. Marrow. Core specification and experiments in DIET: A decentralised ecosystem-inspired mobile agent system. In *Proceedings of the 1st International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS2002)*, pages 623–630, Bologna, Italy, July 2002.
4. D.G.A. Mobach, B.J. Overeinder, O. Marin, and F.M.T. Brazier. Lease-based decentralized resource management in open multi-agent systems. In *Proceedings of the 18th International FLAIRS Conference*, Clearwater Beach, FL, May 2005.
5. B.C. Neuman. Scale in distributed systems. In T. Casavant and M. Singhal, editors, *Readings in Distributed Computing Systems*, pages 463–489. IEEE Computer Society Press, Los Alamitos, CA, 1994.
6. N.J.E. Wijngaards, B.J. Overeinder, M. van Steen, and F.M.T. Brazier. Supporting Internet-scale multi-agent systems. *Data Knowledge Engineering*, 41(2–3):229–245, 2002.
7. M. J. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

Automatic Generation of Wrapper Code and Test Scripts for Problem Solving Environments

Andreas Schreiber

Deutsches Zentrum für Luft- und Raumfahrt e.V., Simulation and Software Technology
Linder Höhe, 51147 Cologne, Germany

Abstract. Problem Solving Environments are useful tools to support complex simulation tasks in the field of engineering and research (e. g. aerospace, automotive, etc.). The TENT environment represents this type of system, into which external applications are being integrated. In this paper two software engineering issues are addressed, which are dealing with the automation of specific tasks while developing and extending the TENT system.
First, we will deal with the automatic generation of code for integrating new applications into the system. This is important for reducing the effort of the manual creation of new application components. Code generation is implemented as a plug-in for the open source development platform Eclipse.
Second, the automatic test case generation and execution for system tests. This feature of test generation is embedded within the TENT graphical user interface. It records all user actions as Python scripts to be used for automatic replaying the operations as a set using the embedded Python interpreter.

1 Introduction

In complex scientific problem solving environments (PSEs) usually lots of modules are integrated into to the system to extend its functionality and to cover broader problem classes [1,2]. In many PSEs this integration of new modules (e. g., new numerical methods) is a software development task. For example, to extend a PSE with an external application one has to develop some kind of wrapper or interface code to connects the application to the programming and extension interfaces of the PSE.

Usually the software engineering tasks for developing software systems consist of several phases: Analysis, design, coding, unit testing, integration testing, and system testing. Depending on the software development process, the order and frequency of these phases may vary, but almost all development processes have two major primary objectives in common [3]: To avoid errors and to find undiscovered errors as early as possible.

For extending PSEs these are the two major software engineering phases: the actual development of the code for wrapping the new modules, and the test of the extendend PSE. Usually the analysis and design phases are less important, because the overall design of the PSE should be finished and the (programming) interfaces for the PSE extension should already be specified.

For PSEs the necessity to extend it with new modules occurs relatively often. Therefore one has got to develop some kind of wrapper code frequently. These codes are usually quite similar, but one still has to test the extended PSE and check for correct results. To support these tasks, tools or methods can be used to automate parts of the process. The major objectives for these supporting tools or methods are: To reduce the effort and to minimize coding errors.

A feasible technique to accomplish these objectives is the automated generation of code. Especially in our case the codes for wrapping new modules and scripts for running test cases can be generated quite easily.

In the following we give a very short overview of the integration system TENT [4] for creating domain specific problem solving environments. After that we first present how new applications are integrated into the system using code generation, and how this is achieved with the help of a plug-in for the development environment Eclipse [5]. The second topic is the automatic generation of test scripts in Python [6] which are used to test the TENT system using an embbeded Python interpreter.

2 Integration System TENT

TENT is a software integration and workflow management system that helps to build and manage process chains for complex simulations in distributed environments. All tools typical for scientific simulation workflows can be integrated into TENT, and controlled from the user's desktop using a graphical user interface (GUI, see Fig. 1). The key features of TENT are:

– Flexible configuration, online steering, and visualization of simulations.
– Utilization of distributed computing resources such as PCs, workstations, clusters, supercomputers, and computational grids.
– Integration of a project based data management with support for cooperative working.

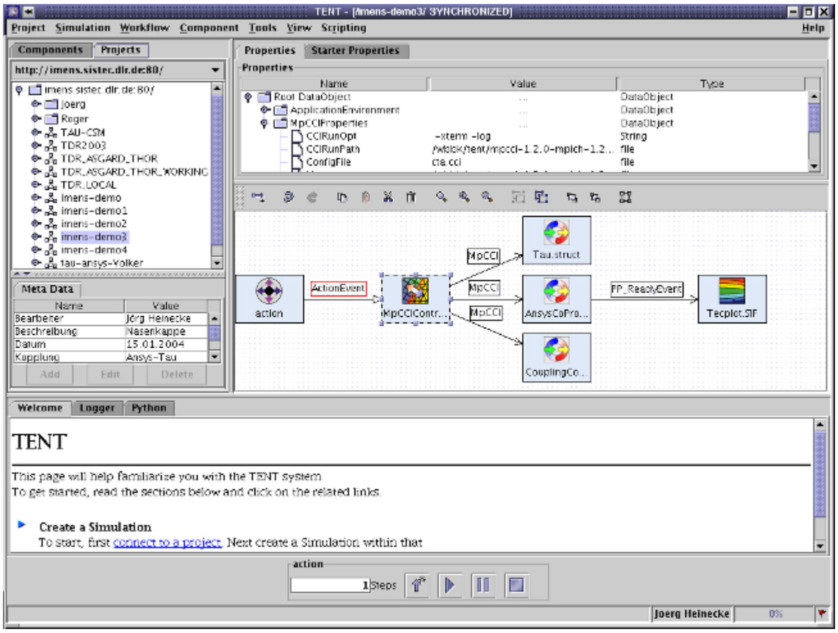
Additional features are:

– Visual composition of process chains (workflows).
– Monitoring and steering of simulations from any computer on the internet.
– Easy integration of existing applications.
– Automatic and efficient data transfer between the stages in the process chain.

The TENT system has been developed as a component system using several software technologies as Java, CORBA [7], the Globus Toolkit [8], XML, Python, and WebDAV [9]. Fig. 2 shows the general architecture of the TENT system. See [4] for more details.

2.1 TENT Components

Components in TENT are defined by interfaces specified in CORBA IDL (Interface Definition Language). All TENT components are inherited from a common base interface which is extended with component specific methods and event interfaces. Events are the primary means of communication between TENT components; they are defined as IDL



Name	Value	Type
Root DataObject	...	DataObject
ApplicationEnvironment	...	DataObject
MpCCIProperties	...	DataObject
CCIRunOpt	-xterm -log	String
CCIRunPath	/wlclk/tent/mpcci-1.2.0-mpich-1.2...	file
ConfigFile	cta.cci	file

Name	Value
Bearbeiter	Jörg Heinecke
Beschreibung	Nasenkappe
Datum	15.01.2004
Kopplung	Ansys-Tau

TENT

This page will help familiarize you with the TENT system.
To get started, read the sections below and click on the related links.

Create a Simulation
To start, first connect to a project. Next create a Simulation within that

Fig. 1. TENT Graphical User Interface

data structures in combination with corresponding listener interfaces. Components can be connected within workflows by connecting one component's suitable event sender to another component's corresponding event listener.

To integrate new applications into TENT, one has to provide a TENT component (wrapper) for the particular application. The implementation of a component can be divided into the following steps:

1. Definition of the component's interfaces by defining its IDL interfaces (inherited from a base interface, inclusion of event listener interfaces, and definition of methods for steps 3 and 4).
2. Implementation of the component's interfaces (inherited from the generated CORBA skeleton base classes) and initialization of properties and data transfer.
3. Development of application-related communication code (depends on desired functionality of the component).
4. Development of necessary GUI plug-ins (depends on controllability of the component).
5. Creation of a proper default configuration file for the component's set of properties.

2.2 TENT-Based Problem Solving Environments

The integration framework TENT is being used to build specific problem solving environments for solving high-level domain tasks. To get the PSE, all necessary applications need to be integrated into TENT. The number of applications varies for each PSE, it

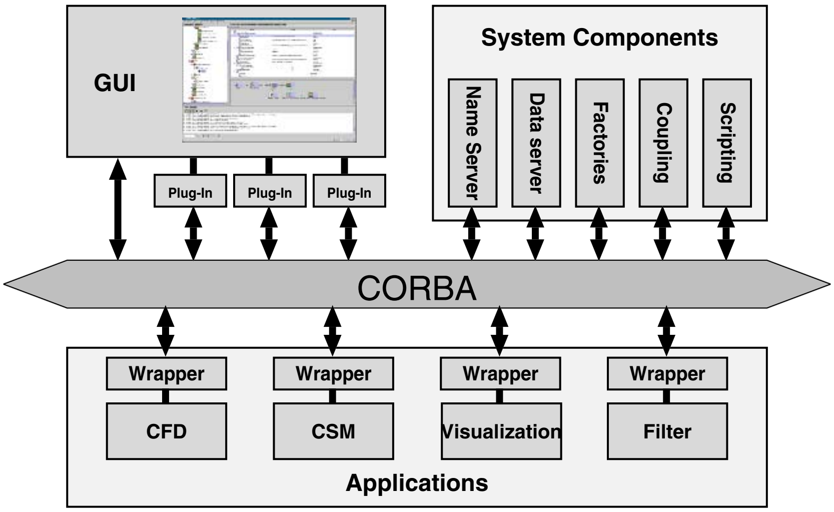


Fig. 2. Architecture of the TENT system

ranges from just a few numerical codes to many dozens of codes together with analysis, pre-/post-processing, and visualization tools.

PSEs based on the TENT framework have been created for aerodynamic studies of civil aircrafts [10], numerical analysis of combat aircraft maneuvering [11,12], numerical simulation of space re-entry vehicles [13,14], or constructional optimization and crash simulation of car bodies [15].

3 Code Generation for Extending PSEs

Our goal was to obtain effective tools for extending TENT-based PSEs with new components according to the steps described in section 2.1. The chosen approach was the development of tools based on Eclipse in the form of plug-ins to enhance the code creation efficiency. The most important one is the plug-in for the automatic generation of TENT component code.

3.1 What Is the Eclipse Platform?

"The Eclipse Platform is designed for anything, but nothing in particular" [16]. It is a universal open platform for tool integration. The basic framework of Eclipse is very extensible via plug-ins (like Wizards, Editor, or Views). For example, the standard Eclipse distribution is distributed with a set of plug-ins for Java development which makes Eclipse a very competetive Java IDE (Integrated Development Environment).

3.2 Generating TENT Components Using Eclipse

To generate TENT components (wrapper codes), we have developed a set of Eclipse plug-ins consisting of wizards and editor extensions. In the wizard all details for the component are specified. The editor extension's purpose is to add or change certain code features in the generated (or already existing) code. Additionaly to components, the code generation wizard can create code for TENT GUI plug-ins.

The creation of a TENT component includes the definition of the component IDL interface and their Java implementation according to steps one and two in section 2.1. The user defines meta data (as the name and location of the generated code) interactively using Wizard dialogs. A dialog (see Fig. 4) provides the means to select the listener interfaces for the events to receive. This information will be added to the component's IDL interface and the Java implementation code. The base for the code to generate is specified in template files (see Fig. 3). After merging all metadata and additional code with the templates, the result is a functional basic TENT component code for wrapping applications. Currently, these still have to be edited manually to add non-standard code functionalities. But this remaining work is well supported by the standard Eclipse *Tasks* view. It presents a list of all places within the code which need manual change or completion (see Fig. 4).

The tool for component code generation described here is an Eclipse *Feature*. An Eclipse feature consists of more than one plug-in and can be used to extend the basic Eclipse platform easily. Our Eclipse feature consists of two plug-ins: The first plug-in consists of the component code generation functionality, the second contains an implementation of the Velocity template engine [17].

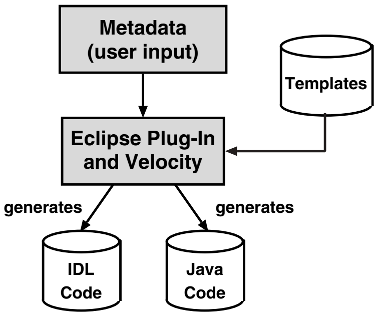


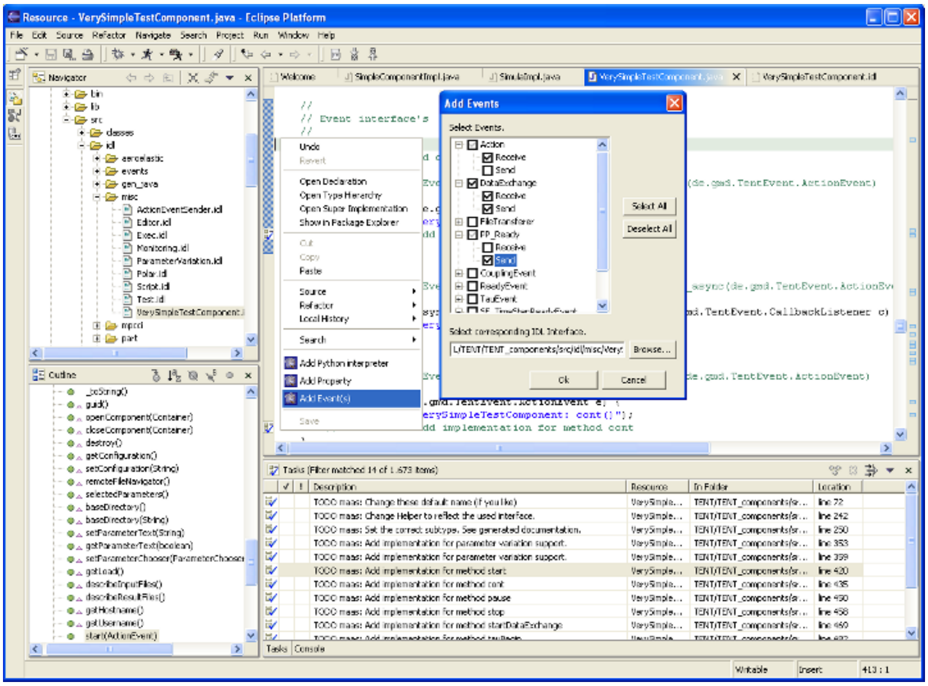
Fig. 3. Generation of wrapper code using templates

3.3 Current Implementation

Velocity is used for the generation of the TENT component source code. It is a template engine implemented in Java and can be used to generate web pages, Java code, or any plain text based output from templates. The templates give the possibility to use declarations, conditional statements or loops. After getting all necessary variables, the engine interprets the template and replaces the variables inside.

The wizard for creating new components, and the dialogs that are opened from within the Java source code editor (Fig. 4), are implemented using the Standard Widget Toolkit (SWT) [18].

The component code generation plug-in described uses the `org.eclipse.ui.newWizards` extension point. After installation the wizard can



Tasks (Filter matched 14 of 1,673 items)

✓	!	Description	Resource	In Folder	Location
		TODO maas: Change these default name (if you like)	VerySimple...	TENT/TENT_components/sr...	line 72
		TODO maas: Change Helper to reflect the used interface.	VerySimple...	TENT/TENT_components/sr...	line 242
		TODO maas: Set the correct subtype. See generated documentation.	VerySimple...	TENT/TENT_components/sr...	line 250
		TODO maas: Add implementation for parameter variation support.	VerySimple...	TENT/TENT_components/sr...	line 353
		TODO maas: Add implementation for parameter variation support.	VerySimple...	TENT/TENT_components/sr...	line 359
		TODO maas: Add implementation for method start	VerySimple...	TENT/TENT_components/sr...	line 420
		TODO maas: Add implementation for method cont	VerySimple...	TENT/TENT_components/sr...	line 435
		TODO maas: Add implementation for method pause	VerySimple...	TENT/TENT_components/sr...	line 450
		TODO maas: Add implementation for method stop	VerySimple...	TENT/TENT_components/sr...	line 458
		TODO maas: Add implementation for method startDataExchange	VerySimple...	TENT/TENT_components/sr...	line 460

Fig. 4. Eclipse with generated Java code, popup menu with extensions, and dialog for adding events

be found using *New* → *TENT* → *Wrapper Component* (see Fig. 5). All available wizards in Eclipse are accessible at this point. So creating a new TENT component with Eclipse is similar to starting any (general) new Java project.

4 Script Generation for System Tests

4.1 System Testing

The system test is a major test phase. It tests the system as a whole. The main goal of the system test is the verification of the overall functionality (see [19] or [20] for more details on software tests). For a PSE that means overall testing of the base framework together with all integrated modules.

During the system test many things need testing. Some very common tests are:

- Functional tests (including GUI tests),
- performance and stress tests,
- client/server tests,
- usability tests, and
- installation tests.

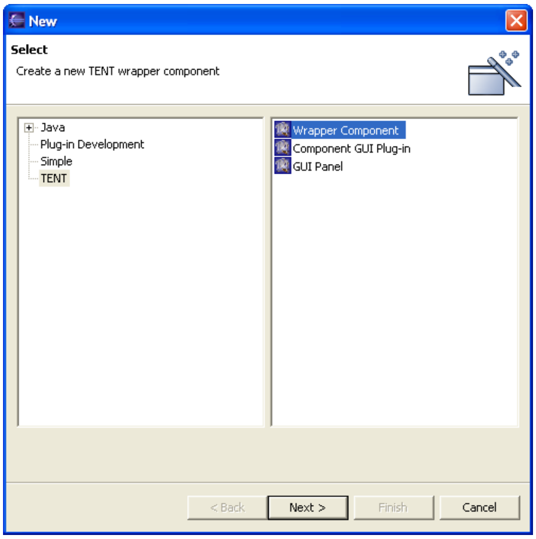


Fig. 5. Wizard for new TENT specific classes

To be effective, the system test has to be performed frequently. On the other hand, performing these tests by humans (developers or quality assurance staff members) is relatively expensive and slow. A more appropriate way for performing system tests is the creation of proper test scripts, especially for the functional, performance, stress and client/server tests. These test can then be automated and started manually or automatically.

In the TENT development the system test were formerly done by a quality assurance staff member who performed test procedures specified in written documents. The test results were added to the document and – in case of discovered errors – in a bug tracking system. Some GUI tests have been performed using a commercial capture/replay tool (*TestComplete* [21]).

4.2 Script Based System Testing in TENT

Currently all manual tests are being replaced by automated scripted tests for performing complex system tests. In general there are two different methods for creating test scripts: The first method is the manual coding of these scripts. The major disadvantage of coding them by hand is that this approach is slow and error-prone. The second method is the automatic generation or recording of these scripts by code generators or the application itself. This method is much faster and the scripts are error-free.

Our technique for generating test scripts is *Command Journaling*. Originaly, Command Journaling has been added to TENT for tracing user actions (an audit trail feature to answer the question "What did I do?"). In Command Journaling all user actions are logged to a *journal* or *session file* in a format that can be replayed or executed afterwards. In TENT the actions the user is performing in the GUI are logged in a Python syntax.

For this purpose all important Java classes of the TENT GUI are recording all actions as Python code to a journal file, which can be used to replay the same operations by a Python interpreter. In TENT, the generated journal script can be run in the GUI embedded Python interpreter (Jython [22,23]), or non-interactively in a stand-alone Python interpreter without the need of a GUI (which is useful for batch processing jobs).

The following Python code shows an example of a generated journal script. In this example, the user added four components to the workflow editor, connected them by events, set some parameters, and started a simulation run.

```
# Command journaling, generated example script (Python syntax)
from de.dlr.tent.gui import GUI
cf = GUI.getControlFacade()
cf.addComponent("IDL:de.dlr.tent/ActionEventSender:1.0.Type*ActionEvent.Subtype*action.Instance")
cf.addComponent("IDL:de.dlr.tent/Script:1.0.Type*Script.Subtype*script.Instance")
cf.addComponent("IDL:de.dlr.tent/Polar:1.0.Type*PolarControl.Subtype*PolarControl.Instance")
cf.addComponent("IDL:de.dlr.tent/Simulation:1.0.Type*FLOWer.Subtype*AGARD.Instance")
cf.addWire("IDL:de.dlr.tent/ActionEventSender:1.0.Type*ActionEvent.Subtype*action.Instance",
    "IDL:de.dlr.tent/Script:1.0.Type*Script.Subtype*script.Instance", "ActionEvent")
cf.addWire("IDL:de.dlr.tent/Script:1.0.Type*Script.Subtype*script.Instance",
    "IDL:de.dlr.tent/Polar:1.0.Type*PolarControl.Subtype*PolarControl.AGARD.Instance",
    "ActionEvent")
cf.addWire("IDL:de.dlr.tent/Script:1.0.Type*Script.Subtype*script.Instance",
    "IDL:de.dlr.tent/Simulation:1.0.Type*FLOWer.Subtype*AGARD.Instance",
    "Simulation")
cf.addWire("IDL:de.dlr.tent/Script:1.0.Type*Script.Subtype*script.Instance",
    "IDL:de.dlr.tent/Simulation:1.0.Type*FLOWer.Subtype*AGARD.Instance",
    "Simulation")
cf.activateAll()
cf.getComponent("IDL:de.dlr.tent/Simulation:1.0.Type*FLOWer.Subtype*AGARD.Instance").
    setProperty("Executable", "/work/flower116")
cf.saveWorkflowAs("polar_with_FLOWer.wfl")
cf.getComponent("IDL:de.dlr.tent/ActionEventSender:1.0.Type*ActionEvent.Subtype*action.Instance").
    fireStart(1)
cf.getComponent("IDL:de.dlr.tent/Polar:1.0.Type*PolarControl.Subtype*PolarControl.Instance").
    setDoubleProperty("EndValue", 40)
cf.getComponent("IDL:de.dlr.tent/Polar:1.0.Type*PolarControl.Subtype*PolarControl.Instance").
    setDoubleProperty("Increment", 2)
```

In practice, the TENT system tester has to perform the following steps to create and run system test cases:

1. The tester performs all steps to test a certain functionality using the GUI. The result is a generated journal script.
2. If necessary, the tester edits the generated script. This can be necessary for removing unneeded commands, for adding evaluation statements to check the internal state of TENT for errors, for generalizing file path names, or for extending the script with further and more complex action, such as loops.
3. Optionally, the new test script may be added to a test suite.
4. The tester starts the test run by invoking the test script using either the embedded Python interpreter of the TENT GUI or a stand-alone Python interpreter.
5. All discovered errors are reported to the bug tracking system. This is is currently still a manual task, but the future goal is to automate this step as well, so that the generated test script submits a new bug tracking entry in case of certain errors.

Most of these steps can also be performed using (commercial) capture/replay tools for GUI tests. But an advantage for creating tests with the described technique is the ability to get information about the running system from within the TENT framework. This is prossible, because the Python interpreter can directly access all public information within the same process. Using this internal information, one can add very sophisticated error detection functionality to the test script.

5 Conclusions

Code generation is more and more important for repetetive and errer-prone tasks in all kinds of software development. As we described, code generation can be a very useful technique in scientific computing and related topics. It can streamline the process of creating and debugging code. In our opinion, developers will be writing code generators instead of writing "regular" code more often in the future.

In the presented solution for generating wrapper code, there is still some manual work necessarry for specific and unforseen functionalities. But the goal is the complete generation of wrapper code. It should be possible to integrate new methods into PSEs simply by entering some meta information about the code or application to be added. As a base for developing a tool for extending our integration framework TENT, the open source platform Eclipse has been used. We think that Eclipse is a well suited development framework for all types of GUI-based applications and tools (see also [24]).

Acknowledgements

The author would like to acknowledge the work of Thijs Metsch (BA Mannheim and DLR) for his implementation of the first Eclipse code generation plug-in.

References

1. J. R. Rice and R. F. Boisvert, *From scientific software libraries to problem-solving environments*, IEEE Computational Science and Engineering, Fall, pp. 44–53, 1996.
2. S. Gallopoulos, E. Houstis, and J. R. Rice, *Problem-solving environments for computational Science*, IEEE Computational Science and Engineering, Summer, pp. 11–23, 1994.
3. I. Sommerville, *Software Engineering*, Addison-Wesley, Harlow, UK, 5th edition, 1995.
4. A. Schreiber, *The Integrated Simulation Environment TENT,* Concurrency and Computation: Practice and Experience, Volume 14, Issue 13–15, pp. 1553–1568, 2002.
5. Eclipse home page. http://www.eclipse.org
6. Python Language Website. http://www.python.org
7. CORBA home page. http://www.omg.org/corba
8. Globus Project Website. http://www.globus.org
9. J. Whitehead and M. Wiggins, *WebDAV: IETF Standard for Collaborative Authoring on the Web,* In IEEE Internet Computing, Vol 2, No. 5, Sept/Oct, 1998.
10. R. Heinrich, R. Ahrem, G. Guenther, H.-P. Kersken, W. Krueger, J. Neumann, *Aeroelastic Computation Using the AMANDA Simulation Environment*. In Proc. of CEAS Conference on Multidisciplinary Design and Optimization (DGLR-Bericht 2001–05), June 25–26, 2001, Cologne, pp. 19–30
11. Project SikMa home page. http://www.dlr.de/as/forschung/projekte/sikma
12. A. Schütte, G. Einarsson, A. Madrane, B. Schöning, W. Mönnich, and W.-R. Krüger, *Numerical Simulation of Manoeuvring Aircraft by Aerodynamic and Flight-Mechanic Coupling*, RTA/AVT Symposium on Reduction of Military Vehicle Aquisition Time and Cost through Advanced Modeling and Virtual Product Simulation, Paris, 2002, RTO-MP-089, RTO/NATO 2003.
13. A. Schreiber, T. Metsch, and H.-P. Kersken, *A Problem Solving Environment for Multidisciplinary Coupled Simulations in Computational Grids*, In Future Generation Computer Systems, in press, 2005.

14. R. Schäfer, A. Mack, B. Esser, A. Gülhan, *Fluid-Structure Interaction on a generic Model of a Reentry Vehicle Nosecap*. In Proc. of the 5th International Congress on Thermal Stresses, Blacksburg, Virginia 2003.
15. Project AUTO-OPT home page. http://www.auto-opt.de
16. Eclipse Platform Technical Overview, Object Technology International, Inc., February 2003.
17. Velocity Template Engine. http://jakarta.apache.org/velocity
18. Standard Widget Toolkit. http://www.eclipse.org/platform/index.html
19. R. J. Patton, *Software Testing*, Sams Publishing, 2000.
20. D. Graham and M. Fewster, *Software Test Automation. Effective Use of Test Execution Tools*, Addison–Wesley, 1999.
21. TestComplete product home page. http://www.automatedqa.com/products/tc.asp
22. Jython home page. http://www.jython.org
23. S. Pedroni, N. Rappin, *Jython Essentials*, O'Reilly and Associates, 2002.
24. T. E. Williams and M. R. Erickson, *Eclipse & General-Purpose Applications*, Dr. Dobbs Journal, September 2004, pp. 66–69.

Runtime Software Techniques for Enhancing High-Performance Applications: An introduction

Masha Sosonkina

Ames Laboratory and Iowa State University
Ames IA 50010, USA
masha@scl.ameslab.gov

Preface

Parallel computing platforms advance rapidly, both in speed and in size. However, often only a fraction of the peak hardware performance is achieved by high-performance scientific applications. The main reason is in the mismatch between the way parallel computation and communication are built into applications and the processor, memory, and interconnection architectures, which vary greatly. One way to cope with the changeability of hardware is to start creating applications able to adapt themselves "on-the-fly". In accomplishing this goal, the following general questions need to be answered: How does an application perceive the system changes? How are the adaptations initiated? What is the nature of application adaptations? The papers in this minisymposium attempt to answer these questions by providing either an application-centric or a system-centric viewpoint. Given changing computational resources or a variety of computing platforms, application performance may be enhanced by, for example, modifying the underlining computational algorithm or by using "external" runtime tools which may aid in better load balancing or mapping of scientific applications to parallel processors. A new dynamic load balancing algorithm is considered by Dixon for particle simulations. Cai shows a way to decrease parallel overhead due to compute node duplication in parallel linear system solution. For a parallel adaptive finite element method implemented on clusters of SMPs, Hippold et al. investigate the interaction of cache effects, communication costs and load balancing. Argollo et al. take a system-centric approach and propose a novel system architecture for geographically distributed clusters. This architecture is tested for a matrix multiplication application. Sosonkina integrates an external middleware tool with a parallel linear system solver to monitor its communications and to inform about possible times to invoke adaptive mechanisms of the application.

Efficient Execution of Scientific Computation on Geographically Distributed Clusters

Eduardo Argollo[1], Dolores Rexachs[1], Fernando G. Tinetti[2], and Emilio Luque[1]

[1] Computer Science Department, Universitat Autònoma de Barcelona, Spain
eduardo.argollo@aomail.uab.es,
{dolores.rexachs,emilio.luque}@uab.es
[2] Fac. de Informática, Inv. Asistente CICPBA, Universidad Nacional de La Plata, Argentina
fernando@info.unlp.edu.ar

Abstract. To achieve data intensive computation, the joining of geographically distributed heterogeneous clusters of workstations through the Internet can be an inexpensive approach. To obtain effective collaboration in such a collection of clusters, overcoming processors and networks heterogeneity, a system architecture was defined. This architecture and a model able to predict application performance and to help its design is described. The matrix multiplication algorithm is used as a benchmark and experiments are conducted over two geographically distributed heterogeneous clusters, one in Brazil and the other in Spain. The model obtained over 90% prediction accuracy in the experiments.

1 Introduction

For an economical approach to achieving data intensive computation, one increasing widespread approach is the use of dedicated heterogeneous networks of workstations (HNOW) with standard software and libraries. These clusters have become popular and are used to solve scientific computing problems in many universities around the world. However, the user's needs are usually beyond the performance of these clusters.

Internet, through its evolution over recent decades, has become a real possibility to interconnecting geographically distributed HNOWs in such a way that the joint behave as a single entity: the collection of HNOWs (CoHNOW). Obtaining effective collaboration of this kind of systems is not a trivial matter [1].

In a CoHNOW there are two levels of communication: an intra-cluster network level to communicate machines locally, and an inter-cluster network level, responsible for interconnecting the clusters. To achieve an efficient level of collaboration performance between the clusters the use of efficient policies relating to workload distribution is crucial. This need is increased when Internet is used, due to its unpredictable latency and throughput, and its performance limitations.

A key concept is to evaluate the possibilities of using the CoHNOW as a single cluster with some specific characteristics. To do this evaluation, this paper proposes a system architecture, a system model and an application tuning methodology.

The system architecture evolved from the initial proposal of [2] in such a way that the CoHNOW has been organized as a hierarchical Master/Worker-based collection of clusters. In each communication level, pipeline strategies are implemented in such a way that communication and computation can be overlapped. For intra-cluster communications MPI MPICH library [3] is used and to optimize the inter-cluster communication performance, guarantee transparency and add reliability to Internet connections, a specific service was introduced to the architecture: the Communication Manager.

Some of the characteristics of the multiple-cluster system (worker performance, intra and inter network throughput) and of the algorithm (workload size and its distribution management) were examined to obtain an analytical system model that permits the prediction of the execution performance and explains its behavior over time. The proposed methodology describes how to assess whether the collaboration makes sense and, if it does, it then describes how to tune the application, adjusting some of the parameters, thereby identifying the best strategies for obtaining the best performance.

In order to demonstrate and evaluate the proposed architecture, methodology and data distribution policies in the field of scientific computation, it is necessary to select a representative application as the benchmark program.

The matrix multiplication (MM) problem, using a blocked algorithm [4], is the selected benchmark application because it is a highly scalable problem with an easily manageable workload, including the possibility of granularities changes. In addiction, the MM operation is a key component of the Linear Algebra Kernel [5] [6] used by a wide range of scientific applications. Its execution over different-speed processors turns out to be surprisingly difficult. Actually, its NP-completeness was proved [7].

Experiments were done in order to validate our work using two geographically distributed clusters: one located in Brazil and the other in Spain. Each cluster is a dedicated HNOW and they are interconnected by the Internet.

The following sections present our study in further detail. Section 2 briefly describes the system architecture. The analytical model for the system and benchmark application is explained in section 3. In section 4, experiments description, obtained results and details are described and results are shown. Finally, conclusions and further work are presented in section 5.

2 System Architecture

The necessity of constantly controlling network usability and ensuring a dynamic rearrangement of the workload of each HNOW led us to use an extended and hierarchical version of the Master/Worker programming paradigm (M/W) as the run-time system model.

The proposed hierarchical organization, see Fig.1, implies there is a "main-cluster", from which the application execution starts, connected to the "remote clusters" by the communication manager (CM). These "remote clusters", or sub-clusters, are organized around their sub-masters and sub-workers.

Two different strategies were developed for the two different communication levels. Inside the local network, its low latency and high throughput permits the utilization of standard MPI library functions. To manage the inter-cluster communication task, carried out over a long-distance network (LDN), a specific service has been developed to be included in each cluster: the Communication Manager (CM).

CMs isolate the local network from the public one, manage the public network disconnections, guarantee the inter-cluster communication and maintain it continuously,

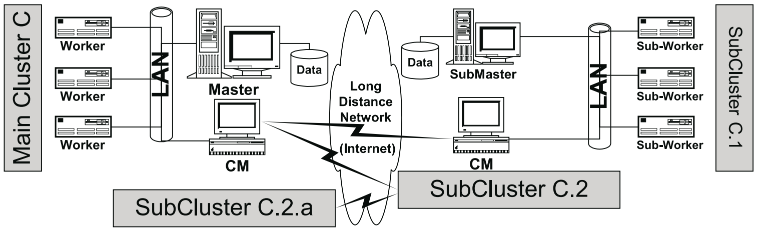


Fig. 1. System architecture as a hierarchical Master/Worker CoHNOW

exploiting as much as possible the unpredictable communication link throughput and bandwidth peaks by means of an adaptive multithreading communication scheme [8].

3 System Model

In such a heterogeneous system, the CoHNOW, it is important to determine if the collaboration is profitable for a certain execution. It is also desirable to have, in advance and along the application execution, a performance estimation, including the expected execution time. To attain these estimation goals an analytic model of the system was developed.

The purpose of this model is to evaluate the possibilities of an efficient collaboration on a CoHNOW, taking into consideration the characteristics and parameters of system's clusters, intra and inter communication links and application. When the collaboration makes sense, the model equations predict the application execution behavior and its performance. The model can also be used in the development of a methodology to design and tune the application in order to obtain the best possible performance over the available computational resources.

The system model is based in a simple computation-communication analysis that is applied to each communication level: inside a cluster (intra-cluster) and between a pair of clusters (inter-cluster). It is important to remark that this approach can be applied for any amount of interconnected clusters, with any possible hierarchical M/W topology since, in this scheme, one cluster just communicates with one other cluster.

In this section we explain the generic computation-communication analysis in both communication levels: intra and inter-cluster. We apply this analysis to the selected benchmark program, the matrix multiplication (MM) algorithm.

3.1 Computation-Communication Analysis

The computation-communication analysis is oriented to the evaluation of the possible performance that can be obtained as a function of the communication network throughput. This evaluation is done through the comparison of the amount of data needed to be communicated for a certain workload, and the amount of operations involved in processing this workload.

For a certain workload, the "computation time" (*CptTime*) can be defined as the ratio between the number of operations (*Oper*) and the system performance (*Perf*). The communication time (*CommTime*) is the ratio between the volume of data communication (*Comm*) and the interconnection network throughput (*TPut*). Using the pipeline strategy, there will be no workers' idle time whenever "communication time" is equal-to "computation time", this means *maximum system efficiency* is achieved. We can then conclude that the attainable system performance depends on the network throughput as shown in Eq. 1.

$$CommTime = CptTime \Rightarrow Perf = \frac{Oper * TPut}{Comm} \tag{3.1}$$

The performance that can be obtained is then the minimum between the Available System Performance (*ASP*) and the performance limited by the network (*Perf*). The Available System Performance (*ASP*) can be obtained experimentally and represents the summarization of the performance for each worker executing the algorithm locally (without communication).

This simple analysis should be applied to the target algorithm, for each workload distribution level (intra and inter-cluster), providing the prediction of the overall execution performance.

3.2 Intra-cluster Performance Contribution Evaluation

At the intra-cluster level, the MM blocked algorithm will be used so that the master will divide the M x M elements matrix in blocks of B x B elements. The master sends a pair of matrix blocks to be multiplied, receiving one result block. The amount of communication (*Comm*) is then the total number of bytes of the above mentioned three blocks, *Comm*=$3 * \alpha * B^2$, being (α) the floating point data size in bytes. For this workload, the number of operations (floating point multiplications and additions) is *Oper*=$2B^3 - B^2$.

Applying the same computation-communication approach of section 3.1, considering now the local area network throughput (*LanTPut*), the local Cluster Performance Limit (*CPL*) in flops is provided by Eq. 2. This equation determines the performance limits for the intra-cluster execution and it can be used to determine the best local block granularity with which the best local execution performance can be obtained.

$$CPL = \frac{(2B^3 - B^2) * LanTPut}{3 * \alpha * B^2} = \frac{(2 * B - 1) * LanTPut}{3 * \alpha} \tag{3.2}$$

The Expected Cluster Performance (*ECP*) is then the minimum between its *CPL* and the experimentally obtained Available System Performance (*ASP*) of the cluster.

3.3 Inter-cluster Performance Contribution Evaluation

Considering a cluster as a single element we can then analyze the inter-cluster communication and the workload distribution and management between clusters in order to determine the amount of the remote clusters performance that can be attained to the CoHNOW.

The maximum CoHNOW performance is obtained when idle time is avoided on its clusters. To do this for the remote clusters, despite the low LDN throughput, a different workload granularity size should be used. For some algorithms this distribution can be improved using the data locality in a way that the granularity will have a dynamic growth since the previous received data can be reused when joined with the recently received data.

This improvement can be obtained for the blocked MM algorithm through a pipeline-based distribution of complete first-operand rows of blocks (R) and second-operand columns of blocks (C). Each time a new row/column (R/C) pair reaches the sub-master, it is added to the previously received rows and columns, exploiting the O(n^3) computation complexity against the O(n^2) data communication complexity of the MM algorithm, therefore increasing the computation-communication ratio.

Using the whole row by column multiplication operations, *Oper*=$2MB^2 - B^2$, and the result communication, *Comm*=$\alpha * B^2$, in the computation-communication analysis it is possible to derive the equation that describes the Potential Contribution Performance Limit (*CoPeL*) the remote cluster can provide as a function of the LDN average throughput (*LdnAvgTPut*) and the matrix elements (Eq. 3).

$$CoPeL = \frac{(2MB^2 - B^2) * LdnAvgTPut}{\alpha * B^2} = \frac{(2 * M - 1) * LdnAvgTPut}{\alpha} \tag{3.3}$$

Then, for a remote cluster, the Expected Performance Contribution (*EPC*) is the minimum value between the *CoPeL* and the cluster *CPL*.

In the inter-cluster pipeline, when a new pair P of R/C is received, *Comm*=$2 * \alpha * M * B$, then 2*P -1 new row by column blocks multiplications operations are possible, *Oper*=$(2P-1)(2MB^2-B^2)$. This application property implies a constant growth in the remote cluster performance collaboration, until the Expected Performance Collaboration (*EPC*) value is reached. This moment is called stabilization point because, from this time on, the contributed performance value will stabilize.

Substituting in Eq. 1 *Comm* and *Oper* for the inter-cluster pipeline, and deducing P, Eq. 4 is obtained. The value of P represents the number of block rows and columns to be sent to reach the stabilization point. Consequently the *StabilizationTime*, the elapsed execution time until this stabilization point is reached, in other words the time spent to send P rows and columns blocks, is provided by Eq. 5.

It is important to note that although the inter-cluster throughput is variable, its average value can be easily obtained experimentally. Based on this average value the availability and performance of the collaboration can be established and the equations can be dynamically calculated through those parameter variations so that workload and flow actions can be taken in order to maintain or achieve new levels of collaboration.

$$P = \frac{\alpha * M * ContribPerf}{(2 * M * B - B) * LdnAvgPut} + \frac{1}{2} \tag{3.4}$$

$$StabilizationTime = \frac{P * Comm}{LdnAvgPut} = \frac{2 * P * \alpha * M * B}{LdnAvgPut} \tag{3.5}$$

4 Experiments

In order to check the accuracy of the developed model, experiments were executed over the testbed CoHNOW. Some experiments results will be shown and one experiment will be analyzed in more detail. Table 1 is presenting the obtained results for 4 significant experiments. The experiments duration are in the range of 400 to 4,000 minutes in order to cover the Internet variable behavior along the time. The prediction for all experiments present over 90% precision.

The experiments were chosen with two different granularities: 100 x 100 block size was chosen to illustrate the capability of prediction with the local area networks saturated, while 400 x 400 block size was obtained with the use of the model equations because it represents the smallest block on which the best execution performance is possible.

To evaluate the Internet throughput contribution, two different CM usage levels were selected to obtain averages throughput of 10KB/sec and 50KB/sec. These averages values were empirically evaluated after exhaustive monitoring of the network behavior.

Table 1. Prediction and real execution comparison for different experiments

	Single CM Thread		Multiple CM Thread	
Matrix Order (M)	10,000	20,000	10,000	10,000
Block Order (B)	100	400	100	400
Expected LdnAvgTput (KB/sec)	10	10	50	50
Predicted Values				
Brazil Cluster Perf (MFlops)	16.78	28.97	16.78	28.97
Spain Cluster Perf (MFlops)	16.78	58.10	16.78	58.10
Stabilization R/C (P)	34	29	7	6
Stabilization Time (min)	433	3,007	18	64
Execution Time (min)	1,106	4,117	998	409
Experiments Results				
CoHNOW Performance (MFlops)	32.72	79.02	31.90	78.80
Stabilization R/C (P)	34	29	7	6
Execution Time (min)	1,145	4,482	1,038	448
Prediction Accuracy	96.4%	91.8%	96.2%	91.3%

The performance behavior along time for the 10,000 x 10,000 matrix using 400 x 400 block with 50KB/sec of expected Internet bandwidth experiment is shown in Fig.2. For the same experiment, Fig.3 shows the real LDN link throughput along the time.

The performance attained by the remote cluster ("remote" line in Fig. 2) increases until it reaches the stabilization point at minute 64. From this time on the cluster is contributing with its best performance. Although at minute 130 there was an abrupt drop of performance (**a** in Fig. 2) because of a 5 minutes Internet disconnection (**a** in Fig. 3).

This disconnection caused no penalty for the overall system performance since the pipe was full at that time and after the reconnection all results of the disconnected period were sent, causing a sudden peak of performance (**b** in Fig. 2).

The model equations were also used to determine when to stop sending data to the remote cluster (**b** in Fig. 3) because it has enough work data for the remaining of the predicted execution time.

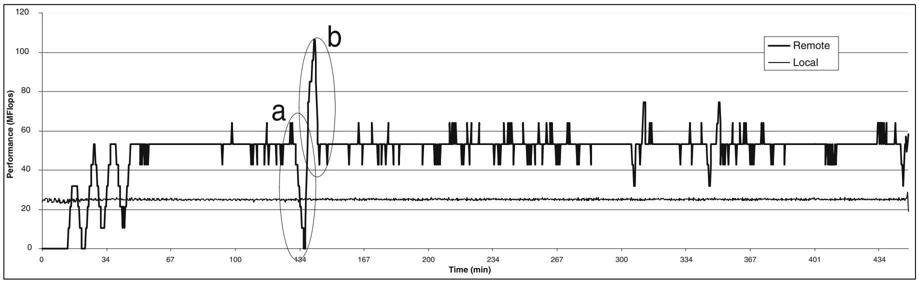


Fig. 2. Experiment M=10000 B=400 LdnAvgTput = 50KB/s: execution performance behavior

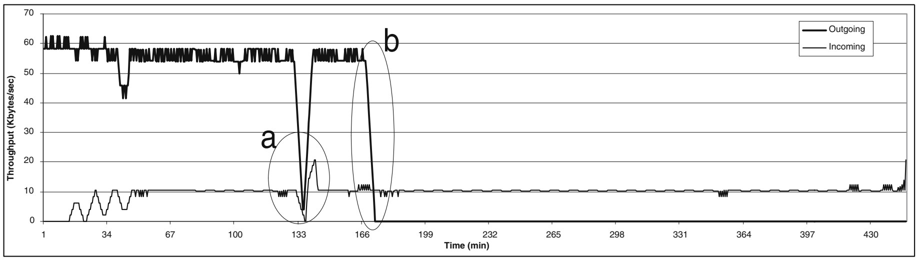


Fig. 3. Experiment M=10000 B=400 LdnAvgTput = 50KB/s: communication throughput along the time

5 Conclusions and Future Work

In order to evaluate the possibilities using a CoHNOW as a "single cluster", a system architecture was defined including a new service, the CM that is in charge of managing the long-distance communication and a system model was developed to predict and tune the application execution. Experiments to validate the model predicted results were made.

For the selected benchmark application, some experiments, with different sizes, granularities and inter-cluster network throughputs, were made and the obtained results had been predicted with an error less than 9%. The performance behavior for a specific experiment was analyzed and this analysis demonstrates that the architecture support the inter-cluster effective collaboration and that the model can be used to predict the stabilization point and the performances behavior before and after it.

Some important future lines consist of generalizing the model to include other scientific computation applications, and the selection of the optimal CoHNOW virtual configuration for maximizing the global system performance.

Acknowledgment

The authors want to thank the anonymous reviewers whose comments led to a substantial clarification of the contribution of this paper.

References

1. Olivier Beaumont, Arnaud Legrand, and Yves Robert. The master-slave paradigm with heterogeneous processors. *IEEE Trans. Parallel Distributed Systems*, 14(9) : 897–908, 2003.
2. A. Furtado, J. Souza, A. Rebouças, D. Rexachs, E. Luque. Architectures for an Efficient Application Execution in a Collection of HNOWS. In D. Kranzlmüller et al. editors, *Proceedings of the 9th Euro PVM/MPI 2002*, Lecture Notes in Computer Science vol. 2474, pages 450–460, 2002.
3. W. Gropp, E. Lusk, N.Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard *Scientific and Engineering Computation Series*, Parallel Computing vol 22 number 6, 789–828, 1996.
4. M. S. Lam, E. Rothberg, M. E. Wolf. The Cache Performance and Optimizations of Blocked Algorithms. *4th Intern. Conference on Architectural Support for Programming Languages and Operating Systems*, Palo Alto CA, pp 63-74, April 1999.
5. Dongarra J., J. Du Croz, S. Hammarling, I. Duff. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, 1990.
6. Dongarra J., D. Walker. Libraries for Linear Algebra. in Sabot G. W. (Ed.), *High Performance Computing: Problem Solving with Parallel and Vector Architectures*, Addison-Wesley Publishing Company, Inc., pp. 93–134, 1995.
7. Beaumont O., F. Rastello and Y. Robert. Matrix Multiplication on Heterogeneous Platforms. *IEEE Trans. on Parallel and Distributed Systems*, vol. 12, No. 10, pp. 1033-1051, October 2001.
8. E. Argollo, J. R. de Souza, D. Rexachs, and E. Luque. Efficient Execution on Long-Distance Geographically Distributed Dedicated Clusters. In D. Kranzlmüller et al. editors, *Proceedings of the 11th Euro PVM/MPI 2004*, Lecture Notes in Computer Science vol. 3241, pages 311–319, 2004.

Improving the Performance of Large-Scale Unstructured PDE Applications

Xing Cai[1,2]

[1] Simula Research Laboratory, P.O. Box 134, N-1325 Lysaker, Norway
xingca@simula.no
[2] Department of Informatics, University of Oslo
P.O. Box 1080 Blindern, N-0316 Oslo, Norway

Abstract. This paper investigates two types of overhead due to duplicated local computations, which are frequently encountered in the parallel software of overlapping domain decomposition methods. To remove the duplication-induced overhead, we propose a parallel scheme that disjointly re-distributes the overlapping mesh points among irregularly shaped subdomains. The essence is to replace the duplicated local computations by an increased volume of the inter-processor communication. Since the number of inter-processor messages remains the same, the bandwidth consumption by an increased number of data values can often be justified by the removal of a considerably larger number of floating-point operations and irregular memory accesses in unstructured applications. Obtainable gain in the resulting parallel performance is demonstrated by numerical experiments.

1 Introduction

An important class of domain decomposition (DD) algorithms is based on the Schwarz iterations that use *overlapping* subdomains; see e.g. [3,8]. These algorithms have become popular choices for solving partial differential equations (PDEs) mainly due to three reasons: (1) a simple algorithmic structure, (2) rapid numerical convergence, and (3) straightforward applicability for parallel computers. The series of the international DD conferences has shown, among other things, the wide spread of the overlapping DD algorithms; see e.g. [1,6,5]. Moreover, the overlapping DD approach is well suited for implementing parallel PDE software, especially with respect to code re-use. This is because the involved subdomain problems are of the same type as the target PDE in the global domain, and no special interface solvers are required. Therefore, an existing serial PDE solver can, in principle, be re-used as the subdomain solver inside a generic DD software framework; see [2]. Besides, the required collaboration between the subdomains, in the form of communication and synchronization, can be implemented as *generic* libraries independent of the target PDE.

The starting point of an overlapping DD method is a set of overlapping subdomains that decomposes the global solution domain as $\Omega = \cup_{s=1}^{P} \Omega_s$. Figure 1 shows such an example of partitioning an unstructured finite element mesh, where a zone of overlapping points is shared between each pair of neighboring subdomains. Roughly speaking, each subdomain is an independent "working unit" with its own discretization task and a subdomain solver. Collaboration between the subdomains is through the exchange

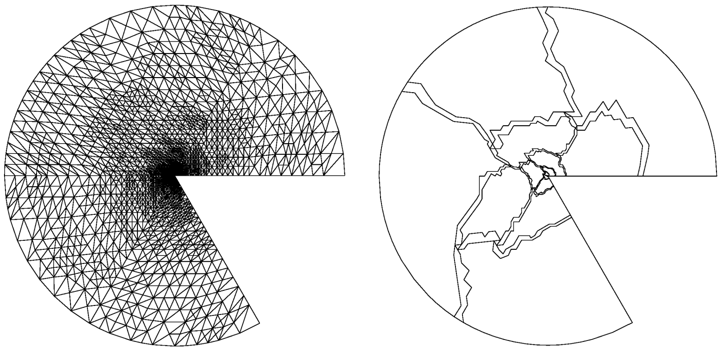


Fig. 1. An example of an unstructured computational mesh and one possible partitioning into several overlapping and irregularly shaped subdomains

of local solutions inside the overlapping zones. A light-weight "global administrator" is required to iteratively improve the "patched" global solution and to synchronize the working pace of all the subdomains. We remark that sufficient overlap between neighboring subdomains is important for obtaining the rapid convergence of an overlapping DD method; see e.g. [3,8].

The main computational tasks involved in an overlapping DD method are in the form of linear algebra operations, most importantly, matrix-vector product, inner-product between two vectors, and addition of vectors. These linear algebra operations are carried out on two levels: the global level and the subdomain level. The global-level operations are typically needed to check the global convergence of the DD method. In particular, when the overlapping DD method is used as a preconditioner (see [3,8]), such as to obtain more robust convergence of a global Krylov solver, global-level operations also constitute the computational kernel of the Krylov solver. In such a setting, the original global system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is replaced by an equivalent system $\boldsymbol{B}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{B}\boldsymbol{b}$. For example, an additive Schwarz preconditioner is defined as

$$\boldsymbol{B}^{\text{add}} = \sum_{s=1}^{P} \boldsymbol{R}_s^T \boldsymbol{A}_s^{-1} \boldsymbol{R}_s. \tag{1.1}$$

In the above definition, $\boldsymbol{A}_s^{-1}$ refers to a local solver within Ω_s (see Section 2), while matrices $\boldsymbol{R}_s^T$ and $\boldsymbol{R}_s$ denote, respectively, the associated interpolation and restriction operators. For more mathematical details, we refer to [3,8]. The computational kernel of each Schwarz iteration, such as (1.1), is constituted by the subdomain-level operations which typically find an approximate local solution on each subdomain, when a suitable subdomain right-hand side is given.

For each overlapping mesh point that is shared between multiple host subdomains, a global-level operation should give rise to the *same* value in all its host subdomains, whereas a subdomain-level operation may compute *different* values in the different host subdomains. Therefore, duplicated computations arise in the global-level operations, provided that the parallel software is based on a straightforward re-use of existing serial linear algebra software over all the local points on each subdomain. Removing the duplication-induced overhead associated with parallel matrix-vector products and inner-products thus constitutes the main theme of the remaining text. In Section 2, a distributed data structure is explained and the associated duplicated computations are examined in detail in Section 3. The main idea of a disjoint re-distribution of all the mesh points is presented in Section 4, and we propose in Section 5 a parallel scheme for carrying out such a disjoint re-distribution. The issue of re-ordering the mesh points on each subdomain is also explained. Finally, Section 6 presents numerical experiments demonstrating the obtainable improvement of parallel performance, after the duplicated local computations are removed from global-level parallel matrix-vector products and inner-products.

2 Distributed Data for Linear Algebra Operations

In the context of parallel overlapping DD methods, the involved linear algebra operations use a *distributed* data structure. That is, all the mesh points (including the overlapping points) in a subdomain participate in constructing the local matrix and vectors of the subdomain. In addition to being used in the subdomain-level operations, such a distributed set of subdomain matrices and vectors is also used in global-level operations. This is because a global matrix or vector can be collectively represented by the set of subdomain local matrices or vectors.

A global matrix $\boldsymbol{A}$ is thus represented by a set of subdomain matrices $\boldsymbol{A}_s$, $1 \leq s \leq P$, where P denotes the number of subdomains and $\boldsymbol{A}_s$ arises from discretizing the target PDE restricted inside Ω_s. We note that some of the rows in $\boldsymbol{A}$ are duplicated among neighboring subdomain matrices, due to the overlap between subdomains. Here, it is important to note the so-called *internal boundary* of Ω_s, which refers to $\partial\Omega_s \backslash \partial\Omega$, i.e., the part of $\partial\Omega_s$ that borders into a neighboring subdomain (thus not on the global boundary $\partial\Omega$). Since a Dirichlet type boundary condition is normally applied on the internal boundary of Ω_s, the corresponding rows in $\boldsymbol{A}_s$ are different from those in $\boldsymbol{A}$, whereas the other rows in $\boldsymbol{A}_s$ are identical with their corresponding rows in $\boldsymbol{A}$.

A global solution vector, say $\boldsymbol{x}$, is distributed as the subdomain vectors $\boldsymbol{x}_s$, $1 \leq s \leq P$. The "overlapping entries" of $\boldsymbol{x}$ are assumed to be always correctly duplicated on all the neighboring subdomains, including on the internal boundaries. During each Schwarz iteration, the neighboring subdomains may compute different local solutions in the overlapping regions. The subdomains thus need to exchange values between the neighbors and agree upon an averaged value for each overlapping point.

3 Overhead due to Duplicated Computations

Parallel execution of global-level operations can often be realizable through serial local operations plus appropriate inter-subdomain communication. To parallelize a global-level matrix-vector product $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$, we can first individually compute $\boldsymbol{y}_s = \boldsymbol{A}_s\boldsymbol{x}_s$ on each subdomain. Then, the entries of $\boldsymbol{y}_s$ that correspond to the internal boundary points on Ω_s must rely on neighboring subdomains to "pass over" the correct values. The needed communication is of the form that each pair of neighboring subdomains exchanges an array of values.

Regarding the parallelization of a global-level inner-product $w = \boldsymbol{x} \cdot \boldsymbol{y}$, the strategy is to let each subdomain first compute its contribution before being added up globally. A naive approach is to first individually compute $w_s = \boldsymbol{x}_s \cdot \boldsymbol{y}_s$, by applying existing serial inner-product software to all the points on Ω_s. However, the local result w_s must be adjusted before we can add them up to produce w. More specifically, we find

$$\tilde{w}_s = w_s - \sum_{k \in \mathcal{K}_s} \frac{K_{s,k} - 1}{K_{s,k}} x_{s,k} \times y_{s,k}, \tag{3.2}$$

where $\mathcal{K}_s$ denotes an index set containing the local indices of all the overlapping mesh points on Ω_s. Moreover, $K_{s,k}$ is an integer containing the degree of overlap for an overlapping point, i.e., the total number of subdomains in which this point is duplicated. Finally, the desired global result can be found as $w = \sum_{s=1}^{P} \tilde{w}_s$, through an all-to-all collective communication.

The above strategies for parallelizing global-level operations are particularly attractive with respect to code re-use. This is because the involved subdomain-level operations $\boldsymbol{y}_s = \boldsymbol{A}_s \boldsymbol{x}_s$ and $w_s = \boldsymbol{x}_s \cdot \boldsymbol{y}_s$ can *directly* re-use existing serial software, involving all the mesh points on Ω_s. However, the disadvantage of these simple strategies is the resulting overhead due to duplicated computations. More specifically, some entries of $\boldsymbol{y}_s = \boldsymbol{A}_s \boldsymbol{x}_s$ are also computed on other subdomains. Similarly, the local computation of $w_s = \boldsymbol{x}_s \cdot \boldsymbol{y}_s$ is "over-sized" and the later adjustment of form (3.2) introduces additional overhead.

For three-dimensional finite element computations, in particular, each point in an unstructured mesh normally has direct couplings to about 15–20 other points, meaning that each row in $\boldsymbol{A}$ has approximately 15–20 non-zero entries. (The use of higher-order basis functions will give rise to more non-zero entries.) The sparse matrix $\boldsymbol{A}$ is usually stored in a compressed row format, i.e., a one-dimensional floating-point array `a_entries` containing row-wise all the non-zero entries, together with two one-dimensional integer arrays `i_row` and `j_col`, which register the positions of the non-zero entries. The computation of $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ on row i is typically implemented as

```
y[i] = 0.0;
for (j=i_row[i]; j<i_row[i+1]; j++)
  y[i] += a_entries[j]*x[j_col[j]];
```

It can be observed that the memory access pattern for $\boldsymbol{x}$ is irregular and indirect, of form `x[j_col[j]]`. In other words, the combination of a large number of floating-point operations and irregular memory accesses can make the computation per row quite expensive. This motivates a strategy of removing the duplicated computation per overlapping point by one extra data value in communication, which may be performance-friendly for many unstructured PDE applications. We remark that the number of inter-subdomain messages remains the same, so the increased communication volume does not result in more latency, but only consumes more bandwidth.

4 Disjoint Re-distribution of Computational Points

The first step towards removing the duplication-induced overhead, which is described in Section 3, is to find a disjoint re-distribution of the entries of $\boldsymbol{x}$ and $\boldsymbol{y}$ (and also rows of $\boldsymbol{A}$) among the subdomains. The result of enforcing such a disjoint partitioning is that, during $\boldsymbol{y}_s = \boldsymbol{A}_s \boldsymbol{x}_s$, only a subset of the rows in $\boldsymbol{A}_s$ carries out the computation, so that each entry of the global vector $\boldsymbol{y}$ is computed in exactly one subdomain. Similarly, when computing $\boldsymbol{x}_s \cdot \boldsymbol{y}_s$ on Ω_s, we only go through a subset of the entries in $\boldsymbol{x}_s$ and $\boldsymbol{y}_s$ to obtain a "correctly-sized" w_s value, which can then be used to compute $w = \sum_{s=1}^{P} w_s$ without any extra adjustments of form (3.2).

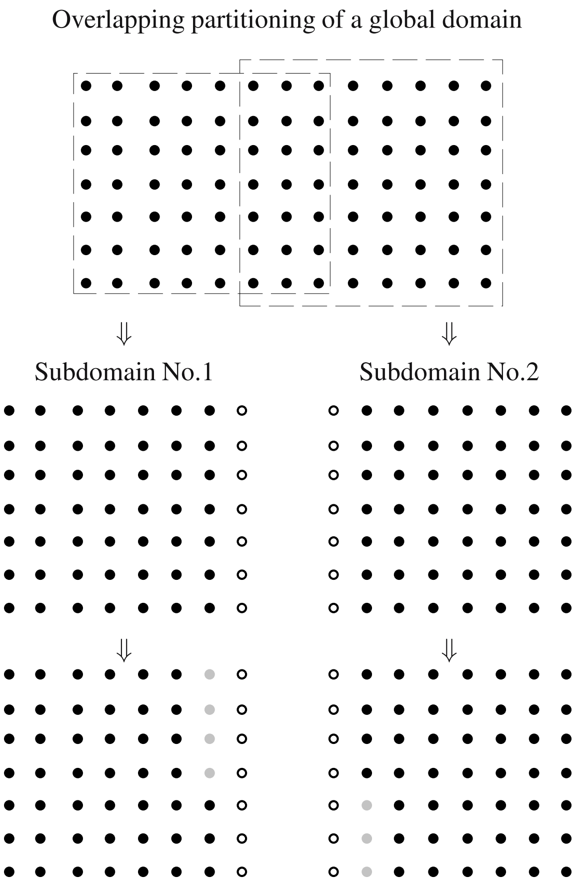


Fig. 2. A simple example of reducing the number of computational points (black-colored) in two overlapping subdomains. The white-colored internal boundary points and the grey-colored overlapping points do not participate in the distributed global-level computations, these points receive computational results from the neighbor

To illustrate the main idea of the disjoint re-distribution, we show a very simple example in Figure 2. In this example, a global domain is decomposed into two overlapping subdomains, with three columns of points in the overlap region. The internal boundary points on each subdomain are marked with the white color. On the middle row of Figure 2, the two columns of overlapping points (those lying immediately beside the internal boundary points) are initially treated on each subdomain as computational points, marked with the same black color as for the interior non-overlapping points. After enforcing a disjoint re-distribution, which is particularly simple for this example, some of the overlapping points are marked with the grey color (see the two subdomains on the bottom row of Figure 2). These grey-colored points thus become "non-computational", meaning that their computational values, together with values of the internal boundary points, are to be provided by the neighboring subdomain. In other words, removal of the duplicated computations comes at the cost of an increased communication volume. We remark that this overhead removal technique only applies to global-level matrix-vector products and inner-products. During each Schwarz iteration, *all* the points on a subdomain have to participate in the subdomain solver.

For unstructured computational meshes, such as that depicted in the left picture of Figure 1, devising a disjoint re-distribution scheme is non-trivial. Moreover, the re-distribution scheme needs to fulfill at least two requirements: (1) it should be based on the existing overlapping partitioning of the points, such that each subdomain chooses a subset of its local points to work as computational points, and (2) the number of computational points per subdomain should be distributed evenly. An optional requirement may be that the re-distribution scheme can be executed in parallel, instead of having to use a "master" processor for doing the re-distribution.

5 A Parallel Re-distribution Scheme

Before presenting a parallel re-distribution scheme, we assume that the global domain Ω has N points and there already exists a set of P overlapping subdomains, where N_s denotes the number of points on Ω_s. Moreover, we assume that $N_s = N_s^I + N_s^O$, where N_s^I denotes the number of interior points on Ω_s, while N_s^O denotes the number of overlapping points on Ω_s, including N_s^B internal boundary points.

A Parallel Re-Distribution Scheme

Phase 1: Mark all the N_s^I interior points on Ω_s as computational points. In addition, some of the non-internal-boundary overlapping points that lie closest to the interior points are also marked, so that the number of initially marked computational points on Ω_s becomes approximately N/P.

A Parallel Re-Distribution Scheme (continued)

Phase 2: Use inter-subdomain communication to find out, for each overlapping point, the number of subdomains (denoted by C) that have initially marked it as a computational point. If $C \neq 1$, choose one host subdomain out of possibly several candidates (in which the considered point does not lie on the internal boundary). Mark thus the overlapping point as a computational point on the host subdomain, and when necessary (if $C > 1$), unmark the overlapping point as non-computational on the other subdomains.

We remark that in the above parallel re-distribution scheme, the choice of the host subdomain in Phase 2 has to be made unanimous among all the candidate host subdomains. For this purpose, we use a unique global number of every overlapping point, say g, which is agreed on all the subdomains beforehand. Let us denote by $S \geq 1$ the number of candidate host subdomains for g. In case none or more than one candidate have initially marked point g, we can select the jth candidate host subdomain, where e.g. $j = \mathrm{mod}(g, S) + 1$. This unanimous selection algorithm has a random nature and preserves the load balance quite satisfactorily, provided that the overlapping subdomains are well balanced originally.

After a re-distribution, all the points on Ω_s receive a “label”, either as a computational point or as a non-computational point. It is clear that all the N_s^I interior points on Ω_s are computational points by default. In addition, some of the overlapping points will also be labelled as computational points. Let us denote by $N_s^{O_c}$ the number of these computational overlapping points on Ω_s (note we always have $N_s^{O_c} \leq N_s^O - N_s^B$). The re-distribution scheme aims at a disjoint partitioning of the N global points among the subdomains, i.e.,

$$N = \sum_{s=1}^{P} \left(N_s^I + N_s^{O_c} \right) .$$

This labeling of the subdomain points is, in principle, sufficient for avoiding overhead due to duplicated computations. However, if the chosen computational points are scattered in the list of all the subdomain points on Ω_s, we have to resort to some kind of `if`-test when traversing the rows of $\boldsymbol{A}_s$ to compute only the desired subset of $\boldsymbol{y}_s = \boldsymbol{A}_s \boldsymbol{x}_s$. The same `if`-test is also needed for traversing the desired subsets of $\boldsymbol{x}_s$ and $\boldsymbol{y}_s$ in association with a parallel inner-product. In practice, this additional `if`-test is a serious obstacle for achieving good computational performance. A remedy to this problem is to re-order the points on each subdomain. More specifically, the first segment contains the N_s^I interior points, and the second segment contains the $N_s^{O_c}$ computational overlapping points. The third segment contains the remaining overlapping points that do not lie on the internal boundary (such as the grey-colored points in Figure 2). Finally, the fourth segment contains the N_s^B internal boundary points.

It is clear that the disjoint re-distribution is unconditionally beneficial to the parallel inner-products, because the number of used entries in $\boldsymbol{x}_s$ and $\boldsymbol{y}_s$ is reduced, while the additional adjustment of form (3.2) is avoided. For the parallel matrix-vector products, the “benefit” on Ω_s is a reduced number of the computational rows from N_s to $N_s^I + N_s^{O_c}$, whereas the “loss” is an increase in the communication volume of order $N_s^O - 2N_s^B$. The latter is because the number of incoming data values is raised from N_s^B to $N_s^O - N_s^{O_c}$, meanwhile the number of outgoing data values is raised from approximately N_s^B to $N_s^{O_c}$. In other words, if the additional bandwidth cost of moving $N_s^O - 2N_s^B$ data values is lower than the computational cost of $N_s^O - N_s^{O_c}$ rows in $\boldsymbol{y}_s = \boldsymbol{A}_s \boldsymbol{x}_s$, we should consider replacing the duplicated computations by an increased communication volume.

6 Numerical Experiments

As a set of concrete numerical experiments, we measure the wall-time consumptions of parallel global-level matrix-vector products and an inner-products, which are associated with the discretization of a Poisson equation on a three-dimensional unstructured mesh using 10,375,680 tetrahedral elements and $N = 1,738,607$ mesh points. The average number of non-zeros per row in the resulting sparse matrix $\boldsymbol{A}$ is approximately 15, and two layers of elements are shared between neighboring subdomains. (The global mesh is partitioned into P overlapping irregularly shaped subdomains by using the Metis package [4] in combination with the parallel software toolbox in Diffpack [7].)

We have tested two quite different parallel platforms, where the first platform is a Linux cluster consisting of 1 GHz Pentium III processors interconnected through a 100 Mbit/s ethernet. The second platform is an SGI Origin 3800 system using 600 MHz R14000 processors. The Linux cluster is particularly interesting because the single-processor computational cost per row in $\boldsymbol{A}$ is approximately the same as the bandwidth cost of moving one data value (8 bytes), i.e., 6.4×10^{-7}s. (We note that the R14000 processors are approximately 60% faster than the Pentium III processors for our unstructured serial computations, whereas the bandwidth advantage of the Origin system over the Linux cluster is more than ten-fold.)

Table 1. The effect of replacing duplicated computations by an increased communication volume; measurements on a Linux cluster and an SGI Origin 3800 system

			Linux cluster				Origin system			
			w/ duplications		w/o duplications		w/ duplications		w/o duplications	
P	$\max N_s$	$\max N_s^C$	T_{matvec}	T_{inner}	T_{matvec}	T_{inner}	T_{matvec}	T_{inner}	T_{matvec}	T_{inner}
2	894605	869385	5.81e-01	4.60e-02	5.77e-01	3.28e-02	4.75e-01	3.50e-02	4.07e-01	2.87e-02
4	465695	435364	3.26e-01	2.63e-02	3.25e-01	2.05e-02	2.69e-01	1.18e-02	2.53e-01	7.06e-03
8	244205	218313	1.79e-01	1.41e-02	1.94e-01	1.06e-02	1.45e-01	6.62e-03	1.32e-01	2.09e-03
16	130156	109550	1.02e-01	1.01e-02	9.75e-02	5.79e-03	8.15e-02	3.07e-03	6.51e-02	1.11e-03
32	69713	55175	N/A	N/A	N/A	N/A	4.09e-02	1.91e-03	3.06e-02	6.58e-04

In Table 1, we denote the wall-time consumption for each matrix-vector product and inner-product by T_{matvec} and T_{inner}, respectively. Moreover, $N_s^C = N_s^I + N_s^{O^c}$ denotes the number of computational points on Ω_s. (The difference between $\max N_s$ and $\max N_s^C$ indicates the amount of removable duplicated computations.) The parallel re-distribution scheme from Section 5 has been applied to duplication removal. The computational cost of the re-distribution itself is quite small, no more than a couple of parallel matrix-vector products. It can be observed from Table 1 that there is always gain in T_{inner} after removing the duplicated local computations. For T_{matvec}, however, the gain on the Linux cluster is smaller than that obtained on the Origin system. This is because the Linux cluster has a much higher bandwidth cost. In one extreme case, i.e., $P = 8$ on the Linux cluster, T_{matvec} has increased. The reason for not obtaining perfect speed-ups, even after removing the duplicated computations, is mainly due to (1) the increased bandwidth consumption and (2) a somewhat imperfect distribution of both $N_s^I + N_s^{O^c}$ and the communication overhead among the subdomains. These factors can be illustrated by Figure 3, where the mesh points on each subdomain are categorized into four types as described in Sections 4 and 5.

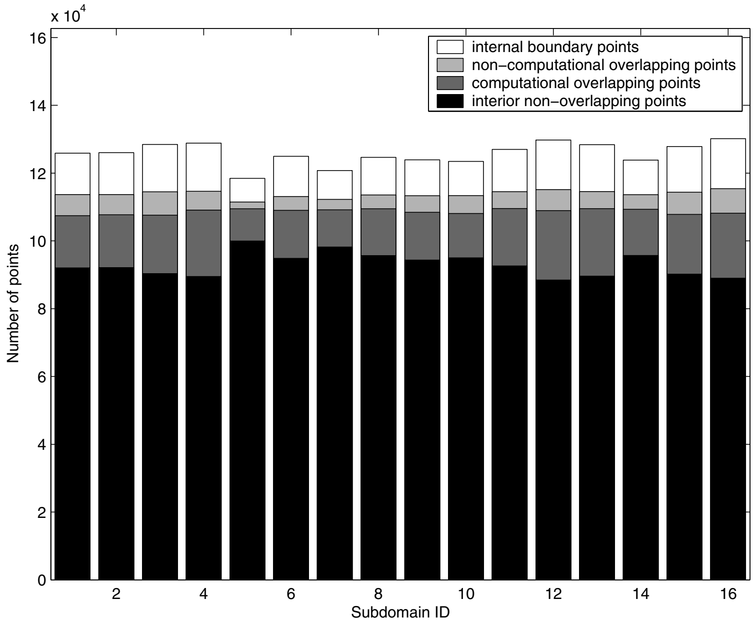


Fig. 3. The categorization of mesh points into four types for sixteen overlapping subdomains

7 Concluding Remarks

In this paper, we have explained the origin of duplicated local computations that frequently arise in parallel overlapping DD software. A disjoint re-distribution of the overlapping mesh points among the subdomains removes the duplications associated with global-level parallel matrix-vector products and inner-products. Although numerical experiments have already shown considerable gain in the parallel performance, more effort is needed to improve the parallel re-distribution scheme from Section 5. It needs first and foremost to produce an even better load balance. Secondly, we may have to "relax" the objective of a complete removal of the duplications in parallel matrix-vector products. That is, a certain amount of duplicated computations may be allowed, such as to limit an acceleration of the communication volume. This means that a "compromise" may be required between reducing the duplicated computations and increasing the communication volume.

Acknowledgement

The work presented in this paper has been partially supported by the Norwegian Research Council (NFR) through *Programme for Supercomputing* in form of a grant of computing time.

References

1. P. E. Bjørstad, M. Espedal, and D. Keyes, editors. *Domain Decomposition Methods in Sciences and Engineering*. Domain Decomposition Press, 1998. Proceedings of the 9th International Conference on Domain Decomposition Methods.
2. X. Cai. Overlapping domain decomposition methods. In H. P. Langtangen and A. Tveito, editors, *Advanced Topics in Computational Partial Differential Equations – Numerical Methods and Diffpack Programming*, pages 57–95. Springer, 2003.
3. T. F. Chan and T. P. Mathew. Domain decomposition algorithms. In *Acta Numerica 1994*, pages 61–143. Cambridge University Press, 1994.
4. G. Karypis and V. Kumar. Metis: Unstructured graph partitioning and sparse matrix ordering system. Technical report, Department of Computer Science, University of Minnesota, Minneapolis/St. Paul, MN, 1995.
5. R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Widlund, and J. Xu, editors. *Domain Decomposition Methods in Sciences and Engineering*, volume 40 of *Lecture Notes in Computational Science and Engineering*. Springer, 2004. Proceedings of the 15th International Conference on Domain Decomposition Methods.
6. C.-H. Lai, P. E. Bjørstad, M. Cross, and O. Widlund, editors. *Domain Decomposition Methods in Sciences and Engineering*. Domain Decomposition Press, 1999. Proceedings of the 11th International Conference on Domain Decomposition Methods.
7. H. P. Langtangen. *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*. Texts in Computational Science and Engineering. Springer, 2nd edition, 2003.
8. B. F. Smith, P. E. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

A Runtime Adaptive Load Balancing Algorithm for Particle Simulations

Matthew F. Dixon[*]

Department of Mathematics, Imperial College, London
matthew.dixon@imperial.ac.uk

Abstract. A new dynamic load balancing algorithm is proposed for particle simulations on clusters of workstations which operates at two levels; it both dynamically defines groups of particles in which the force calculations are localised and at the same time adaptively assigns groups to processors based on the group sizes and runtime CPU usages. It therefore adaptively balances using both the dynamics of the simulation and the load usage of the cluster at runtime.
The algorithm is implemented in the POOMA framework and applied to a particle-in-cell approximation of a three dimensional elastic particle collision model. Load balancing metrics and parallel scalability is determined on an 8 quad-processor 833MHZ Compaq Alpha cluster connected by a gigabit ethernet.

1 Introduction

Particle simulations are widespread in scientific computing and provide a natural computational approach to many equations derived from particle mechanics. particle-in-cell methods remain one of the most common techniques used for particle simulations as they are usually simple. However, like many other particle simulation methods, particle-in-cell methods suffer from poor parallel scalability [8][13]. This is because the force F_i on a particle i with position $\mathbf{X}_i$ is the gradient of the potential energy $V(\mathbf{X}_1, \ldots, \mathbf{X}_N)$ of the particle system (which is model specific)

$$\mathbf{F}_i = -\nabla_{\mathbf{X}_i} V(\mathbf{X}_1, \ldots, \mathbf{X}_N), \ \dot{\mathbf{U}}_i = \frac{\mathbf{F}_i}{m_i} \text{ and } \dot{\mathbf{X}}_i = \mathbf{U}_i$$

depends on the position of the other particles in the system. The velocity $\mathbf{U}_i$ and hence new position of each particle with mass m_i is determined by the above equations of motion.

Parallelisation of the model by assignment of either particles or physical subdomains to different processes leads to either high interprocessor communication requirements or irregular load balance. That is, partitioning of label space into subdomains of equal number of particles, allocated to each processor, does not localise the data required for the force calculations, if particles in close proximity of each other in physical space reside on different processors. See, for example, [9][12]. Conversely, partitioning of the physical domain into subdomains, allocated to each processor, is not necessary compatible with

[*] The author thanks Alvin Chua and Steven Capper for their careful proof reading and useful comments on this article.

the distribution of the work load, namely, calculation of the particle forces. For example, if particles cluster together inside a subdomain, no work load is distributed to the other subdomains.

An extensive number of dynamic load balancing algorithms for parallel particle simulations have been developed which are able to distribute particles across processors based on their proximities to others or by their position in the domain, see for example [2][7][8]. Each consider the main factors in designing an effective load balancing algorithm (i) data locality (ii) granularity of balancing and (iii) the additional computation and interprocessor communication required by the algorithm, referred to hereon, as the overhead. However, none of these approaches use runtime information such as the processor load, performance and latency between other processors to enhance calibration of the algorithm.

In particular, the *taskfarm* approach described in [8], assembles particles into tasks and balances them by instructing processors, which have finished processing their own tasks, to fetch tasks from the others. In theory, this approach could be improved for use on a cluster, by allocating tasks to processors based on processor usages, thus taking a predictive rather than reactive approach. The potential for performance improvement using prediction is dependent on both the level of fluctuation of the usages and the simulated particle dynamics.

Many of the dynamic load balancing schemes that have been developed can be classified as either *Scratch-and-Remap* or *Diffusion-based* schemes [6]. The former approach partitions workload from scratch at each stage whilst the latter achieves balance by successive migration. The choice of scheme is dependent on the dynamics of the simulation. A runtime adaptive algorithm would be most appropriate for a smoothly varying but inhomogeneous particle simulation. Predictive information on the particle clustering can then be used by a diffusion-based load balancing algorithm, to group particles. It would also suit a shared cluster of workstations running multiple applications at any one instance, provided that load usages remain consistent between load balancing cycles and the overhead of rebalancing is small.

A simple and efficient approach for load assignment and rebalance event detection based on measuring efficiency ratios of load to capacity is described in [3]. Since this approach was not developed for particle simulations, there is no concept of balancing at the granularity of dynamically defined groups of entities. This approach is extended, here, to recalibrate particle to processor assignment based on both the particle positions and runtime information. This combined approach, to the best knowledge of the author, has not been previously designed. Different metrics for detecting load balancing events based on the homogeneity of the distribution of the particle are considered. Finally details of the implementation in `POOMA` [4] and numerical results are presented for a parallel 3D elastic particle collision model [1] running on a Compaq Alpha cluster.

2 Algorithm

Consider a single-threaded message-based Single-Processor Multiple-Data parallel model where each processor is represented as a physical node to which numerous virtual nodes (groups) are assigned based on its runtime capacity. Particles are then allocated to each virtual node based on their proximity to one another. Consider first, for simplicity, dynamic balancing of individual particles.

Single Particle Balancing

Let the set of all particles in the system be given by $\mathbf{P} := \{p_1, p_2, \cdots, p_n\}$ where n is the total number of particles in the system. The subset of n_j particles on processor $j \in \{1, 2, \cdots, N_p\}$ is $\mathbf{P}_j(t) := \{p_{\beta_1}, p_{\beta_2}, \cdots, p_{\beta_{n_j}}\}$, using the convention that roman numerals are local indices, such that $\mathbf{P} = \cup_j \mathbf{P}_j$.

Assume that all particles remain equi-distributed and fixed in number, then there is one particle per virtual node and the particles can be considered as virtual nodes. The number of particles assigned to processor j at time t_i, $n_j(t_i)$, depends on n and the weights $w_j(t_i)$. Each $w_j(t_i)$ is a function of the processor peak FLOP rate r_j and its usage $u_j(t_i)$, normalised by the total capacity $C(t_i) = \sum_j C_j(t_i)$.

Particle assignment:

The number of particles initially assigned to each processor is based on its availability and performance.

$$n_j(t_0) = \left\lceil w_j(t_0)n \right\rceil, \; w_j(t_i) = \frac{C_j(t_i)}{C(t_i)} \text{ and } C_j(t_i) = u_j(t_i)r_j.$$

After setting up the system as prescribed, it will be necessary to reassign the particles if the usage of the processor changes.

Particle reassignment:

A processor event causes reassignment if $w_j(t_i) - w_j(t_{i-1}) > w_{tol}$, where w_{tol} is a tolerance to reduce the sensitivity of the load balancing to small changes in relative processor capacity and $t_i - t_{i-1} > \Delta\tau$ the time step size used in the simulation. The number of particles at time t_i is related to the number of particles at time t_{i-1} by the formula

$$n_j(t_i) = \left\lceil \frac{w_j(t_i)}{w_j(t_{i-1})} n_j(t_{i-1}) \right\rceil$$

As such, $n_j(t_i) - n_j(t_{i-1})$ particles are relocated to or from physical node j at time step t_i. Particle relocation, however, reduces parallel scalability of the potential calculations. This motivates the derivation of an algorithm for a courser level of granularity using *groups*.

Group Load Balancing

Multiple particles $p \in \mathbf{P}_j$ can be uniquely assigned to g_j groups $\mathbf{G}_j^{\alpha_l}(t) := \{p_{\gamma_1}, p_{\gamma_2}, \cdots, p_{\gamma_{S_j^{\alpha_l}}}\}$. Defines local subindices $l, m \in \{1, 2, \cdots, g_j\}$ to α on processor j, so that groups are uniquely labelled with the superscript over the computational domain. Let

$X(p_n, t_i)$ be the position of particle p_n at time t_i and $R(.,.)$ the distance between any two particles. If particle proximities to a fictious reference particle $p^{\alpha_l}_{ref}$ are less than a tolerance $r_{tol} = \Delta X(t_0)$, such as the initial equi-interparticle distance, for example, then the particles are assigned to group α_l.

$$\mathbf{G}^{\alpha_m}_j(t_i) := \{p_s \in \mathbf{P}_j,\ p_s \notin \cup_{l<m}\mathbf{G}^{\alpha_l}_j(t_i) | R(X(p_s, t_i), X(p^{\alpha_l}_{ref}, t_i)) < r_{tol}\}$$

$\{p^{\alpha_l}_{ref}\}$ are initially chosen as equidistant points in the Eulerian reference frame and once groups have been assembled are recomputed as the mean position of all the particles in the group. In this way all $S^{\alpha_l}_j$ particles in a group are thus categorised by proximity and are defined as *interacting*. Non-interacting particles are also assigned to single element groups and the load balancing algorithm then calibrates with groups and not particles. For simplicity, it is assumed that the number of particles remain fixed in number.

In addition to defining a metric for load balancing based upon the particles distances and the processor capacities, classification of the particle dynamics can also be used to calibrate the algorithm. For ease of explanation, two categories are considered, (A) weakly and (B) strongly non-equidistributed particle simulations.

Particle Group Assignment

The number of groups initially assigned to each processor j is based on its availability and performance, and mean group size. Consider the two cases A and B of particle classification

$$\text{A: } g_j(t_0) = \left\lceil \frac{w_j(t)n}{\mu(S^m(t_0))} \right\rceil \text{ or B: } g_j(t_0) = \left\lceil \frac{w_j(t)n}{\mu(S^{\alpha_l}_j(t_0))} \right\rceil$$

where the mean group size is $\mu(S^m(t)),\ m \in \{1, 2, \cdots, M\}$ and $M = \sum_j g_j$, and $\mu(S^{\alpha_l}_j)$ is the mean group size on physical node j.

Particle Group Reassignment

The number of groups assigned to processor j at time t_i is related to the number of groups at time t_{i-1} by the formula below. Consider, again, the two cases A and B, where $\hat{\mu} := \mu(S^m)$ and $\mu^* := \mu(S^{\alpha_l}_j)$

$$\text{A: } g_j(t_i) = \left\lceil \frac{\hat{\mu}(t_i)w_j(t_i)}{\hat{\mu}(t_{i-1})w_j(t_{i-1})} g_j(t_{i-1}) \right\rceil \text{ or B: } g_j(t_i) = \left\lceil \frac{\mu^*(t_i)w_j(t_i)}{\mu^*(t_{i-1})w_j(t_{i-1})} g_j(t_{i-1}) \right\rceil$$

For case A reassignment thus depends on the mean size of all groups or, for case B, the mean size of the groups on processor j, depending on the distribution of the particles at time t_i.

The particle distribution is analysed every k^{th} simulation time step and hardware evaluation for context mapping every m^{th} simulation time step, where k and m are suitably chosen parameters based on the dynamics of the particles, available processor capacity and time taken to measure the particle positions and processor state.

3 Implementation

In the POOMA framework, particle indices can be represented as a domain in label space which is partitioned into *patches* (virtual nodes) by *layouts*, which can be easily dynamically reconfigured. One or more patches are then allocated to a context associated with the hardware by a *context mapper*. See [5] for further details.

A special purpose context mapper class has been developed which is initialised by the master node with a set of processor capacities C_i based on a pre-defined map for peak processor capacities whose key is each machine name in the cluster. At each iteration of the load balancing algorithm, each client is instructed to determine the runtime processor statistics by calling the API of the profiling tool on a separate thread. The client then sends this data to the context mapper class on the master node which then determines the number of patches to reallocate to each context sequentially. Finally each client exchanges group data through the master node. This approach is liable to cause a bottleneck at the master processor. A parallel task queue, as used in [10] would help to alleviate this problem.

The clients also exchange particle data peer-to-peer. This is required (i) for a force calculation or (ii) after a repartitioning event. Cross context dependencies can be further minimised by using Guard Layers to store particles of adjacent patches, but this was not used in testing.

Load Balancing Algorithm: Simulation Event

At every k^{th} simulation time step $\tau_{ik} = t_i$ each client j performs the algorithm defined in Figure 3.

Particles on client j are regrouped every k^{th} time step if the total particle displacements, for all local particles, is greater than a threshold tol_{sim}. This approach to associating a particle with a group could be improved using , for example, Voronoi cells as described in [9]. Each client then requests a set of reference nodes for all g_{sum} groups $\{p^{\alpha_l}_{ref}\}$ and the corresponding processors where the groups are located $\{pr_{\beta_l}\}$ defined with a global index β. For each local group, each member particle is checked to determine whether it still belongs to the group. If it doesn't, then it is compared against all reference nodes. If it is associated with a remote reference node, then the particle and the target client are added to the arrays $relocateParticles$ and $targetProcessors$ for dispatch. Otherwise, the particle is reassigned to the target local group.

The client maintains an array of all local particles and the groups to which they belong. The master maintains an array associating groups to the clients and arrays for the size and reference particles for each group. In this way, the master is not overloaded with particle information, but is dedicated to group balancing.

Load Balancing Algorithm: Processor Event

At every m^{th} simulation time step $\tau_{im} = t_i$, each processor j performs the algorithm defined in Figure 3.

The master processor first computes the weights for each client. Processing for each client sequentially, if determines whether the change in a client weight $\Delta w_j(t_i)$ exceeds

```
1: if $||\{\Delta X(p_n)\}|| < tol_{sim},\ p_n \in \cup_l \mathbf{G}_j^{\alpha_l}(t_i)$ then
2:     ReceiveFrom($pr_{master}$, $\{p_{ref}^{\alpha_l}\}$, $\{pr_{\beta_l}\}$, $g_{sum}$)
3:     for $l := 1 \rightarrow g_j(t_i)$ do
4:         for $n := 1 \rightarrow S_j^{\alpha_l}(t_i)$ do
5:             if $R(X(p_{\gamma_n}, X(p_{ref}^{\alpha_l})) > r_{tol}$ then
6:                 for $k := 1 \rightarrow g_{sum}$ do
7:                     if $R(X(p_{\gamma_n}), X(p_{ref}^{\alpha_k})) < r_{tol}$ then
8:                         if $pr_{\beta_k}! = pr_j$ then
9:                             relocationParticles.add($p_{\gamma_n}$)
10:                            targetProcessors.add($pr_{\beta_k}$)
11:                        else
12:                            Update group($\mathbf{G}_j^{\alpha_k}(t_i)$, $p_{\gamma_n}$)
13:                            Update group($\mathbf{G}_j^{\alpha_l}(t_i)$, $p_{\gamma_n}$)
14:                        end if
15:                    end if
16:                end for
17:            end if
18:        end for
19:    end for
20:    SendTo(targetProcessors,relocationParticles)
21:    Receive(relocationParticles)
22:    for $r := 1 \rightarrow N\{$relocationParticles$\}$ do
23:        for $l := 1 \rightarrow g_j(t_i)$ do
24:            Update group($\mathbf{G}_j^{\alpha_l}(t_i)$,relocationParticles[r])
25:        end for
26:    end for
27: end if
```

Fig. 1.

a threshold tol_{cpu} and if so calculates the group imbalance , that is the number of groups above or below its capacity, assuming uniform group sizes. It then searches over all unbalanced clients with a greater index to find the client whose imbalance best matches. It then instructs the client with excess numbers of groups to send them to the target client and the group numbers are appropriately modified. Once a client has been rebalanced, group numbers remain fixed until the next load balancing iteration and the remaining unbalanced clients must exchange groups amongst themselves.

Althought this approach is simple, the last clients to be processed may not be as well balanced as those first processed. Alternating direction of processing of the client lists may potentially improve performance. This approach also assumes, for simplicity, that group sizes remain near uniform, either globally or per client depending on the particle dynamics classification. Many approaches, however, also prioritise by load unit size, see for example [8][12]; this approach could be extended to prioritise larger groups. A further extension is to regroup in a way that keeps messages between neighbouring clients, as suggested in [11].

```
1: if $pr_j != pr_{master}$ then
2:     $C_j(t_i)$=GetCapacity()
3:     SendTo($pr_{master}$, $C_j$)
4:     ReceiveFrom($pr_{master}$, $\Delta g_j(t_i)$, $pr_{target}$)
5:     for $l := 1 \rightarrow \Delta g_j(t_i)$ do
6:         relocationGroups.add($\mathbf{G}_j^{\alpha_l}(t_i)$)
7:     end for
8:     SendTo($pr_{target}$, relocationGroups)
9:     Receive(relocationGroups)
10:    for $r := 1 \rightarrow N\{$relocationGroups$\}$ do
11:        for $l := 1 \rightarrow g_j(t_i)$ do
12:            Update groups($\mathbf{G}_j^{\alpha_l}(t_i)$,relocationGroups[r])
13:            Calculate($\mu(\mathbf{G}_j^{\alpha_l}(t_i))$)
14:        end for
15:    end for
16: else
17:    Receive($\{C_k(t_i)\}$)
18:    for $k := 1 \rightarrow N_p$ do
19:        $w_k(t_i) = \frac{C_k(t_i)}{\sum_n C_n(t_i)}$
20:    end for
21:    for $k := 1 \rightarrow N_p$ do
22:        if $|w_k(t_i)| > tol_{cpu}$ then
23:            $\Delta g_k(t_i) = \left\lceil (\frac{\mu(S^m(t_i))w_k(t_i)}{\mu(S^m(t_{i-1}))w_k(t_{i-1})} - 1)g_k(t_{i-1}) \right\rceil$
24:            sign=$\frac{\Delta g_k(t)}{|\Delta g_k(t)|}$
25:            $min_{j'}(\{\Delta g_j(t_i)\}_{j:=k+1 \rightarrow N_p} + \Delta g_k(t_i))$
26:            if sign$< 0$ then
27:                for $l := 1 \rightarrow \Delta g_{j'}(t_i)$ do
28:                    relocationGroups.add($\mathbf{G}_{j'}^{\alpha_l}(t_i)$)
29:                end for
30:                $pr_{target} = k$
31:            else
32:                for $l := 1 \rightarrow \Delta g_{j'}(t_i)$ do
33:                    relocationGroups.add($\mathbf{G}_i^{\alpha_l}(t_i)$)
34:                end for
35:                $pr_{target} = j'$
36:            end if
37:            SendTo($pr_{target}$, relocationGroups)
38:            $g_k(t_i) -= sign * \Delta g_{j'}(t_i)$
39:            $g_{j'}(t_i) + -= sign * \Delta g_{j'}(t_i)$
40:        end if
41:    end for
42: end if
```

Fig. 2.

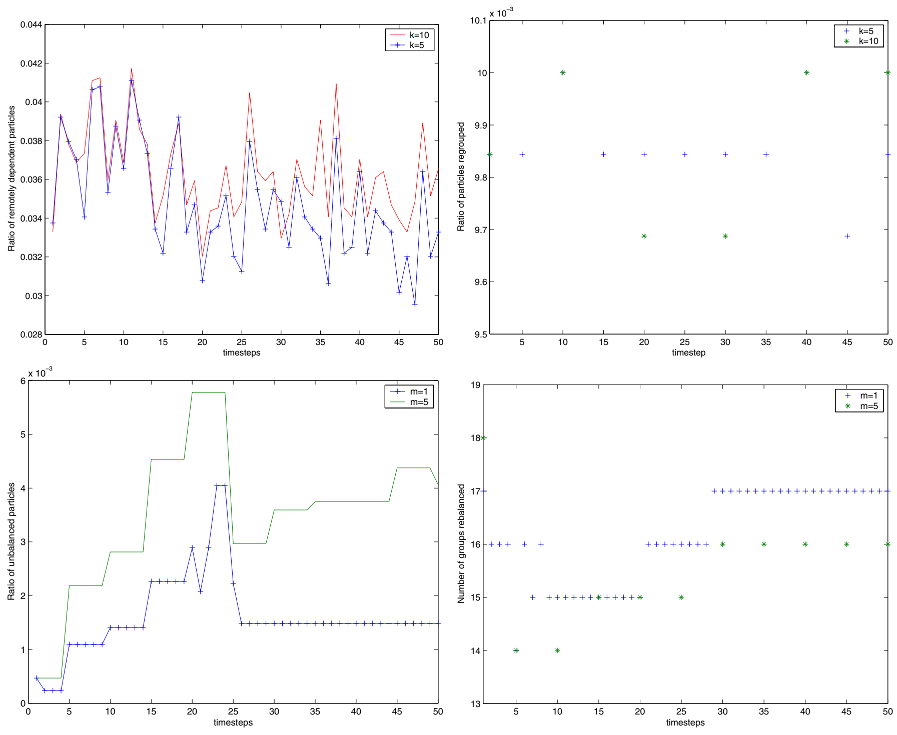


Fig. 3. The effect of k on (top left) remote data dependencies and (top right) regrouping costs. The effect of m on (bottom left) load imbalance and (bottom right) group rebalancing costs

4 Results

The weakly non-equidistributed particle group load balancing algorithm was tested in a particle-in-cell method for a 3D dimensional elastic particle collision model(see [1]) using `POOMA` v2.4.0 and `Cheetah` v1.1.4, a MPI based library built on MPICH with GNU C++ v3.1 on a gigabit ethernet cluster of $8 \quad 4$ x 833 MHZ Compaq Alpha ES40s with Quadrics interconnects running Linux and several applications at any instance. Two sets of results are presented. The first shows the effect of the frequency of particle grouping and rebalancing on the data locality, imbalance and overhead. Parallel scalability is then assessed with and without the dynamic load balancing algorithm.

The group threshold is set to be greater than the radius of influence of the particles. Clearly, this approach requires that the forces are short range otherwise it becomes impossible to construct groups which preserve data locality. The number of particles is chosen to be significantly larger than the number of processors, see [13]. The parameter, tol_{cpu} determines the sensitivity of the load balancing algorithm to load variation. A lower value will ensure more frequent calibration but incur more overhead. Analysis of cpu load histories could be used to determine this parameter. Additionally, m determines the maximum frequency that the load balancing algorithm is performed. It is set to a

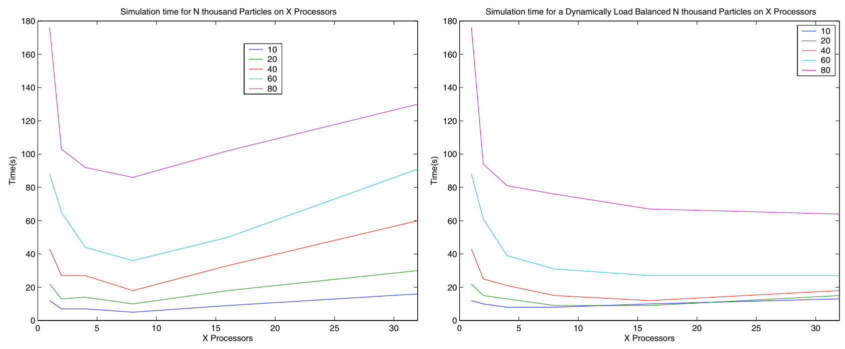


Fig. 4. A 3D elastic particle collision model without (left) and with (right) runtime adaptive load balancing

lower value when the particles are regrouped more frequently, load variations are high and the relative effect of load balancing overhead is small. Figure 3 shows the effect of m on the load imbalance and the corresponding overhead of group balancing for $tol_{cpu} = 0.15$.

The parameter, tol_{sim} sets the sensitivity of the particle regrouping. The value chosen depends on the range of forces, the dynamics of the particles and the effect of remote dependencies on parallel scalability. Additionally, k sets the maximum frequency of particle regrouping and is determined by the particle dynamics. Figure 3 shows the effect of increasing k on the ratio of remotely dependent particles and the corresponding overhead of regrouping for $tol_{sim} = 0.1$. k and m should be compatible with each other; a low value of k results in a potentially high number of particle regrouping and there is thus more potential for a higher frequency of group imbalances.

Parallel scalability tests were performed using 1000 time steps over a range of $10-80$ thousand particles and $1-32$ processors. The load balancing algorithm was performed every $k = m = 100$ time steps. The graphs in Figure 4 demonstrate that the load balancing is much more effective as the number of particles and processors is increased. However, with smaller numbers of particles, the opposite effect is observed because of the overhead of the algorithm.

The greater the number of groups, the greater the granularity of the load balancing but the greater the overhead of rebalancing. In highly latent environments, preliminary results suggest that it is preferable to allocate fewer groups by allocating all non-interacting particles into one group on each processor.

References

1. R.W. Hockney and J.W. Eastwood, *Computer Simulation using Particles*, Taylor and Francis Inc., 1988
2. G.A. Kohring, *Dynamic Load Balancing for Parallelized Particle Simulations on MIMD Computers*, Parallel Computing, Volume 21, Issue 4,pp. 683–694, 1995.
3. J. Watts and S. Taylor, *A Practical Approach to Dynamic Load Balancing*, IEEE Transactions on Parallel and Distributed Systems, Volume 9, issue 3, 1998, pp. 235–248.

4. J.C. Cummings and W. Humphrey, *Parallel Particle Simulations Using the POOMA Framework*, Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Computing, M. Heath et al. eds., SIAM 1997.
5. J.D. Oldham, *POOMA Manual, A C++ Toolkit for High-Performance Parallel Scientific Computation*, `http://www.codesourcery.com`, 2002.
6. Z. Lan, V.E. Taylor and G. Bryan, *A Novel Dynamic Load Balancing Scheme for Parallel Systems*, J. Parallel Distrib. Computing, Volume 62, 2002, pp. 1763–1781.
7. F.R. Pearce and H.M.P. Couchman, *Hydra: a parallel adaptive grid code*, New Astronomy, Volume 2, 1997, pp. 411–427
8. C. Othmer and J. Schüle, *Dynamic load balancing of plasma particle-in-cell simulations: The taskfarm alternative*, Computer Physics Communications, Volume 147, pp. 741–744
9. M. A. Stijnman, R. H. Bisseling and G. T. Barkema, *Partitioning 3D space for parallel many-particle simulations*, Computer Physics Communications, Volume 149, Issue 3, January 2003, pp. 121–134
10. T. Rauber, G. Rünger and C. Scholtes, *Execution behavior analysis and performance prediction for a shared-memory implementation of an irregular particle simulation method*, Simulation Practice and Theory, Volume 6, Issue 7, 15 November 1998, pp. 665–687
11. B. Hendrickson and K. Devine, *Dynamic load balancing in computational mechanics*, Computer Methods in Applied Mechanics and Engineering, Volume 184, Issues 2-4, 14 April 2000, pp. 485–500
12. L. Kalé et al., *NAMD: Greater Scalability for Parallel Molecular Dynamics*, Journal of Computational Physics, Volume 151, 1999, pp. 283–312
13. B. Di Martino et al., *Parallel PIC plasma simulation through particle decomposition techniques*, Parallel Computing, Volume 27, 2001, pp. 295–314

Evaluating Parallel Algorithms for Solving Sylvester-Type Matrix Equations: Direct Transformation-Based Versus Iterative Matrix-Sign-Function-Based Methods

Robert Granat and Bo Kågström

Department of Computing Science and HPC2N, Umeå University
SE-901 87 Umeå, Sweden
{granat,bokg}@cs.umu.se

Abstract. Recent ScaLAPACK-style implementations of the Bartels-Stewart method and the iterative matrix-sign-function-based method for solving continuous-time Sylvester matrix equations are evaluated with respect to generality of use, execution time and accuracy of computed results. The test problems include well-conditioned as well as ill-conditioned Sylvester equations. A method is considered more general if it can effectively solve a larger set of problems. Ill-conditioning is measured with respect to the separation of the two matrices in the Sylvester operator. Experiments carried out on two different distributed memory machines show that the parallel explicitly blocked Bartels-Stewart algorithm can solve more general problems and delivers far more accuracy for ill-conditioned problems. It is also up to four times faster for large enough problems on the most balanced parallel platform (IBM SP), while the parallel iterative algorithm is almost always the fastest of the two on the less balanced platform (HPC2N Linux Super Cluster).

Keywords: Sylvester matrix equation, continuous-time, Bartels–Stewart method, explicit blocking, GEMM-based, level 3 BLAS, matrix sign function, c-stable matrices, Newton iteration, ScaLAPACK, PSLICOT.

1 Introduction

We consider two different methods for solving the continuous-time Sylvester (SYCT) equation

$$AX - XB = C, \qquad (1)$$

where A of size $m \times m$, B of size $n \times n$ and C of size $m \times n$ are arbitrary matrices with real entries. SYCT has a unique solution X of size $m \times n$ if and only if A and B have disjoint spectra, i.e., they have no eigenvalues in common, or equivalently the separation $\mathrm{sep}(A, B) \neq 0$, where $\mathrm{sep}(A, B) = \inf_{\|X\|_F=1} \|AX - XB\|_F$.

The Sylvester equation appears naturally in several applications. Examples include block-diagonalization of a matrix in Schur form and condition estimation of eigenvalue problems (e.g., see [13,11,15]).

In this contribution, we experimentally compare parallel implementations of two methods for solving SYCT on distributed memory systems. The first is based on Bartels-Stewart method [1,15,7,8] and is reviewed in Section 2. Several papers, starting with [16], have considered iterative methods for solving SYCT. The second parallel implementation is an iterative method based on a Newton iteration of the matrix sign function [3,4], and is reviewed in Section 3. In Section 4, we display and compare some computational results of implementations of the two parallel algorithms. A discussion of the measured execution times on two different parallel distributed memory platforms is presented in Section 5. Finally, in Section 6, we summarize our findings and outline ongoing and future work.

2 Explicitly Blocked Algorithms for Solving SYCT

The explicitly blocked method, implemented as the routine `PGESYCTD`, is based on the Bartels-Stewart method [1]:

1. Transform A and B to upper (quasi-)triangular Schur form T_A and T_B, respectively, using orthogonal similarity transformations:
$$Q^T A Q = T_A, \qquad P^T B P = T_B.$$
2. Update the right hand side C of (1) with respect to the transformations done on A and B:
$$\widetilde{C} = Q^T C P.$$
3. Solve the reduced (quasi-)triangular matrix equation:
$$T_A \widetilde{X} - \widetilde{X} T_B = \widetilde{C}.$$
4. Transform the solution $\widetilde{X}$ back to the original coordinate system:
$$X = Q \widetilde{X} P^T.$$

To carry out Step 1 we use a Hessenberg reduction directly followed by the QR-algorithm. The updates in Step 2 and the back-transformation in Step 4 are carried out using general matrix multiply and add (GEMM) operations $C \leftarrow \beta C + \alpha \mathrm{op}(A)\mathrm{op}(B)$, where α and β are scalars and $\mathrm{op}(A)$ denotes A or its transpose A^T [2,12]. We now focus on Step 3. If the matrices A and B are in Schur form, we partition the matrices in SYCT using the blocking factors *mb* and *nb*, respectively. This implies that *mb* is the row-block size and *nb* is the column-block size of the matrices C and X (which overwrites C). By defining $D_a = \lceil m/mb \rceil$ and $D_b = \lceil n/nb \rceil$, (1) can be rewritten as

$$A_{ii} X_{ij} - X_{ij} B_{jj} = C_{ij} - \Big(\sum_{k=i+1}^{D_a} A_{ik} X_{kj} - \sum_{k=1}^{j-1} X_{ik} B_{kj} \Big), \tag{2}$$

where $i = 1, 2, \ldots, D_a$ and $j = 1, 2, \ldots, D_b$. From (2), a serial blocked algorithm can be formulated, see Figure 1.

```
for $j$=1, $D_b$
    for $i$=$D_a$, 1, -1
        {Solve the $(i, j)$th subsystem using a kernel solver}
        $A_{ii}X_{ij} - X_{ij}B_{jj} = C_{ij}$
        for $k$=1, $i - 1$
            {Update block column $j$ of $C$}
            $C_{kj} = C_{kj} - A_{ki}X_{ij}$
        end
        for $k$=$j + 1$, $D_b$
            {Update block row $i$ of $C$}
            $C_{ik} = C_{ik} + X_{ij}B_{jk}$
        end
    end
end
```

Fig. 1. Block algorithm for solving $AX - XB = C$, A and B in upper real Schur form

Assume that the matrices A, B and C are distributed using 2D block-cyclic mapping across a $P_r \times P_c$ processor grid. For Steps 1, 2 and 4, we use the ScaLAPACK library routines PDGEHRD, PDLAHQR and PDGEMM [6]. The first two routines are used in Step 1 to compute the Schur decompositions of A and B (reduction to upper Hessenberg form followed by the parallel QR algorithm [10,9]). PDGEMM is the parallel implementation of the level 3 BLAS GEMM operation and is used in Steps 2 and 4 for doing the two-sided matrix multiply updates.

To carry out Step 3 in parallel, we traverse the matrix C/X along its block diagonals from South-West to North-East, starting in the South-West corner. To be able to compute X_{ij} for different values of i and j, we need A_{ii} and B_{jj} to be owned by the same process that owns C_{ij}. We also need to have the blocks used in the updates of C_{ij} in the right place at the right time. This means that in general we have to communicate for some blocks during the solves and updates. This is done "on demand": whenever a processor misses any block that it needs for solving a node subsystem or doing a GEMM update, it is received from the owner in a single point-to-point communication [8]. A brief outline of a parallel algorithm PTRSYCTD is presented in Figure 2. The (quasi-)triangular subsystems $A_{ii}X_{ij} - X_{ij}B_{jj} = C_{ij}$ in Figure 2 are solved on the grid nodes using the LAPACK-solver DTRSYL [2].

The explicitly blocked method is general since (in theory) it can be used to solve any instance of SYCT as long as the spectra of A and B are disjoint. Notice that the algorithm contains an iterative part, the reduction of A and B in upper Hessenberg form to upper Schur form, which is the most expensive in terms of execution time [7,8]. For further details we refer to [7,8,15].

3 Solving SYCT Using Newton Iteration for the Matrix Sign Function

The sign function method can be used to solve SYCT if the spectra of A and $-B$ are contained in the open left half complex plane, i.e., A and $-B$ are so called *Hurwitz-* or *c*-stable [3].

```
for $k$=1, $D_a + D_b - 1$
    {Solve subsystems on current block diagonal in parallel}
    if(*mynode* holds $C_{ij}$)
      if(*mynode* does not hold $A_{ii}$ and/or $B_{jj}$)
        Communicate for $A_{ii}$ and/or $B_{jj}$
      Solve for $X_{ij}$ in $A_{ii}X_{ij} - X_{ij}B_{ij} = C_{ij}$
      Broadcast $X_{ij}$ to processors that need $X_{ij}$ for updates
    elseif(*mynode* needs $X_{ij}$)
      Receive $X_{ij}$
      if(*mynode* does not hold block in $A$ needed for updating block column $j$)
        Communicate for requested block in $A$
      Update block column $j$ of $C$ in parallel
      if(*mynode* does not hold block in $B$ needed for updating block row $i$)
        Communicate for requested block in $B$
      Update block row $i$ of $C$ in parallel
    endif
end
```

Fig. 2. Parallel block algorithm for $AX - XB = C$, A and B in upper real Schur form

Let Z be a real $p \times p$ matrix with real eigenvalues and let

$$Z = S \begin{bmatrix} J^- & 0 \\ 0 & J^+ \end{bmatrix} S^{-1} \tag{3}$$

denote its Jordan decomposition with $J^- \in C^{k\times k}$, $J^+ \in C^{(p-k)\times(p-k)}$ containing the Jordan blocks corresponding to the eigenvalues in the open left and right half planes, respectively. Then the *matrix sign function* of Z is defined as

$$\operatorname{sign}(Z) = S \begin{bmatrix} -I_k & 0 \\ 0 & I_{p-k} \end{bmatrix} S^{-1}. \tag{4}$$

The sign function can be computed via the Newton iteration for the equation $Z^2 = I$ where the starting point is chosen as Z, i.e.,

$$\begin{aligned} &Z_0 = Z, \\ &Z_{k+1} = (Z_k + Z_k^{-1})/2, \qquad k = 0, 1, 2, \ldots \end{aligned} \tag{5}$$

It can be proved [16] that $\operatorname{sign}(Z) = \lim_{k\to\infty} Z_k$ and moreover that

$$\operatorname{sign}\left(\begin{bmatrix} A & -C \\ 0 & B \end{bmatrix}\right) + I_{m+n} = 2 \begin{bmatrix} 0 & X \\ 0 & I \end{bmatrix}, \tag{6}$$

which means that under the given assumptions, SYCT can be solved by applying the Newton iteration (5) to

$$Z_0 = \begin{bmatrix} A & -C \\ 0 & B \end{bmatrix}. \tag{7}$$

This iterative process only requires basic numerical linear algebra operations as matrix-matrix multiplication, inversion and/or solving linear systems of equations. The method has been parallelized and implemented as the PSLICOT [14,17] routine `psb04md`. The parallel implementation uses the ScaLAPACK routines `PDGETRF` (LU factorization), `PDGETRS` (solving linear systems of equations), `PDGETRI` (inversion based on LU factorization), `PDTRSM` (solution of triangular systems with multiple right-hand sides) and `PDLAPIV` (pivoting of a distributed matrix). We expect this iterative method to be fast and scalable since it consists of computationally well-known and highly parallel operations. The obvious drawback of the method is the lower degree of generality, i.e., the fact that we cannot solve all instances of SYCT due to the restrictions on the spectra of A and B.

The matrix sign function method can also be applied to other related problems, e.g., the stable generalized Lyapunov equation $A^T XE + E^T XA = C$, where $A, E, X, C \in R^{n \times n}$ and $C = C^T$ [3], and it can also be combined with iterative refinement for higher accuracy. However, this has not been incorporated in `psb04md` [4]. For further details we refer to [16,4,3,5].

4 Computational Results

In this section, we present and compare measured speed and accuracy results of `PGESYCTD` and `psb04md` using two different parallel platforms. We solve a number of differently conditioned (regarding the separation of A and B) problems of various sizes using different processor mesh configurations.

Our target machines are the IBM Scalable POWERparallel (SP) system and the Super Linux Cluster at High Performance Computing Center North (HPC2N), where we utilize up to 64 processors of each machine (see Table 4). The test problems are constructed as follows. Consider the matrix A in the form $A = Q(\alpha D_A + \beta M_A)Q^T$, where D_A is (quasi-)diagonal, M_A is strictly upper triangular, Q is orthogonal and α and β are real scalars. We choose M_A as a random matrix with uniformly distributed elements in [-1,1] and prescribe the eigenvalues of A by specifying the elements of D_A. If the matrix B is constructed similarly, we can construct differently conditioned problems by varying the eigenvalues in D_A and D_B and choosing appropriate values of the scaling factors. For example, the factor β is used to control the distance from normality, $\beta\|M_A\|$, of the matrix A.

A representative selection of our results for problems with $m = n$ are shown in Tables 1, 2, and 3. Results for problems with $m \neq n$ are not presented here but will not lead to any different conclusions. We have chosen close to optimal block sizes for the different parallel algorithms and parallel architectures. The upper parts of Tables 1 and 2 correspond to well-conditioned problems, and the lower parts represent moderately to very ill-conditioned problems. We display the performance ratios q_T, q_X and q_R, which correspond to the execution time ratio and two accuracy ratios, the Frobenius norms of the absolute (forward) error $\|X - \tilde{X}\|_F$ and the absolute residual $\|R\|_F = \|C - A\tilde{X} + \tilde{X}B\|_F$ of the two implementations, where X and $\tilde{X}$ denote the exact and computed solutions, respectively. If a ratio is larger than 1.0, `psb04md` shows better results, otherwise `PGESYCTD` is better.

Table 1. Speed and accuracy results on IBM SP system for the routines `PGESYCTD` and `psb04md` solving the general equation $AX - XB = C$ for well- and ill-conditioned problems. All problems use the blocking factors $mb = nb = 64$. In the upper part of the table, A and $-B$ have the eigenvalues $\lambda_i = -i, i = 1, 2, \ldots, m = n$, and $\alpha = \beta = 1.0$. For the lower part A and $-B$ have the eigenvalues: a) $\lambda_{Ai} = -i, \lambda_{Bi} = -1000.0 \cdot i^{-1}, \alpha = \beta = 1.0$, b) $\lambda_{Ai} = -i, \lambda_{Bi} = -2000.0 \cdot i^{-1}, \alpha = \beta = 1.0$, c) A and $-B$ have the eigenvalues $\lambda_i = -i, \alpha_A = \alpha_B = \beta_A = 1.0, \beta_B = 50.0$

			`PGESYCTD`			`psb04md`			Performance ratios		
$m = n$	sep^{-1}	$P_r \times P_c$	T_p	$\Vert X - \tilde{X}\Vert_F$	$\Vert R\Vert_F$	T_p	$\Vert X - \tilde{X}\Vert_F$	$\Vert R\Vert_F$	q_T	q_X	q_R
512	3.7E-3	1×1	60.0	4.1E-12	1.2E-9	43.8	1.1E-12	2.4E-10	1.37	3.73	0.50
512	3.8E-3	2×1	49.2	4.1E-12	1.2E-9	37.1	2.1E-12	9.1E-10	1.33	1.95	1.32
512	3.8E-3	2×2	40.4	4.3E-12	1.2E-9	33.0	1.8E-12	8.0E-10	1.22	2.39	1.50
512	3.8E-3	4×2	33.6	3.8E-12	1.1E-9	33.1	1.9E-12	8.0E-10	1.02	2.00	1.38
1024	2.1E-3	1×1	534	1.2E-11	6.6E-9	529	1.0E-10	3.3E-8	1.01	0.12	0.20
1024	2.1E-3	2×1	287	1.2E-11	6.5E-9	253	8.2E-12	6.8E-9	1.13	1.38	0.96
1024	2.1E-3	2×2	177	1.1E-11	5.4E-9	169	2.2E-11	8.9E-9	1.05	0.50	0.61
1024	2.1E-3	4×2	138	1.1E-11	6.6E-9	170	8.1E-12	6.8E-9	0.81	1.36	0.97
1024	2.1E-3	4×4	106	1.3E-11	6.7E-9	142	6.6E-12	5.5E-9	0.75	1.97	1.22
2048	1.1E-3	2×2	1625	3.8E-11	3.5E-8	6141	2.9E-11	4.9E-8	0.26	1.31	0.71
2048	1.1E-3	4×2	835	3.5E-11	3.7E-8	1031	3.5E-11	4.8E-8	0.81	1.00	0.08
2048	1.1E-3	4×4	446	3.3E-11	3.7E-8	658	2.1E-11	3.6E-8	0.68	1.57	1.03
2048	1.1E-3	8×4	359	4.0E-11	3.7E-8	670	2.5E-11	3.9E-8	0.54	1.60	0.95
2048	1.1E-3	8×8	302	3.5E-11	3.7E-8	588	2.5E-11	3.9E-8	0.51	1.40	0.95
3072	7.6E-4	4×2	3355	6.3E-11	9.7E-8	11504	5.7E-11	1.5E-7	0.29	1.11	0.65
3072	7.6E-4	4×4	1677	6.3E-11	1.0E-7	2473	4.9E-11	1.3E-7	0.68	1.29	0.77
3072	7.6E-4	8×4	1056	6.4E-11	1.0E-7	2078	1.3E-10	2.0E-7	0.51	0.49	0.50
3072	7.6E-4	8×8	705	6.6E-11	1.0E-7	1556	6.3E-10	6.8E-7	0.45	0.10	0.15
4096	5.9E-4	4×4	3788	9.5E-11	2.0E-7	11602	8.3E-11	2.7E-7	0.33	1.14	0.74
4096	5.9E-4	8×4	2330	9.6E-11	2.1E-7	5036	7.8E-11	2.5E-7	0.46	1.23	0.84
4096	5.9E-4	8×8	1365	1.0E-10	2.1E-7	3102	8.8E-11	2.7E-7	0.44	0.11	0.78
512^a	0.17	1×1	61.7	6.1E-11	7.2E-10	33.8	1.1E-9	2.6E-7	1.8	5.5E-2	2.8E-3
512^a	0.16	2×1	47.0	8.5E-11	6.9E-10	29.7	9.4E-10	2.2E-7	1.6	3.1E-3	3.1E-3
512^a	0.15	2×2	36.1	8.8E-11	6.9E-10	26.8	1.2E-9	2.8E-7	1.4	7.3E-2	2.5E-3
512^a	0.16	4×2	30.6	8.3E-11	6.7E-10	25.1	1.1E-9	2.8E-7	1.2	7.5E-2	2.4E-3
1024^b	2.8	1×1	635	7.6E-9	4.0E-9	575	4.9E-5	2.9E-2	1.1	1.6E-4	1.4E-7
1024^b	2.8	2×1	326	4.1E-9	3.9E-9	206	3.1E-5	1.8E-2	1.6	1.3E-4	2.2E-7
1024^b	2.7	2×2	188	8.8E-9	3.8E-9	156	6.0E-5	3.3E-2	1.2	1.5E-4	1.2E-7
1024^b	2.7	4×2	125	1.1E-8	3.8E-9	131	3.6E-5	2.1E-2	0.95	3.1E-4	1.8E-7
1024^b	3.4	4×4	91	9.3E-9	3.8E-9	118	4.4E-5	2.7E-2	0.77	2.1E-4	1.4E-7
2048^c	*mem*	2×2	1797	8.3E-5	4.1E-8	43708	*dnc*	*dnc*	0.0	0.0	0.0
2048^c	4.8E4	4×2	892	5.8E-5	4.1E-8	1158	1.2	1343	0.77	4.8E-5	3.1E-11
2048^c	5.2E4	4×4	446	7.2E-5	4.1E-8	708	0.86	982	0.63	8.4E-5	4.2E-11
2048^c	5.4E4	8×4	348	5.1E-5	4.1E-8	738	0.92	1034	0.47	5.5E-5	4.0E-11
2048^c	4.8E4	8×8	281	4.2E-5	4.1E-8	733	0.88	1003	0.38	4.8E-5	4.1E-11
3072^c	*mem*	4×2	3148	1.4E-5	1.1E-7	11358	2.9	5054	0.28	4.8E-6	2.2E-11
3072^c	3.9E3	4×4	1635	1.1E-5	1.1E-7	2936	1.8	3198	0.51	6.1E-6	3.4E-11
3072^c	4.7E3	8×4	878	7.3E-6	1.1E-7	2051	1.9	3305	0.43	3.8E-6	3.3E-11
3072^c	4.8E3	8×8	769	8.6E-6	1.1E-7	1785	1.7	2926	0.43	5.1E-6	3.8E-11
4096^c	*mem*	4×4	3805	1.9E-4	2.1E-7	15435	85.9	180879	0.25	2.2E-6	1.2E-12
4096^c	5.5E4	8×4	2066	3.3E-4	2.1E-7	41439	*dnc*	*dnc*	0.0	0.0	0.0
4096^c	6.5E4	8×8	1327	1.9E-4	2.1E-7	22487	*dnc*	*dnc*	0.0	0.0	0.0

When the algorithm in `psb04md` converged, it did so in less than 20 iterations. We used an upper threshold of 100 iterations. To signal that `psb04md` does not converge, we use the notation *dnc*.

Recall that SYCT has a unique solution if and only if the A and B have disjoint spectra, or equivalently $\text{sep}(A, B) \neq 0$, where

$$\text{sep}(A, B) = \inf_{\Vert X\Vert_F=1} \Vert AX - XB\Vert_F = \sigma_{\min}(Z_{\text{SYCT}}) = \Vert Z_{\text{SYCT}}^{-1}\Vert_2^{-1}, \quad (8)$$

Table 2. Speed and accuracy results on Super Linux Cluster `seth` for the routines `PGESYCTD` and `psb04md` solving the general equation $AX - XB = C$ for well- and ill-conditioned problems. All problems use the blocking factors $mb = nb = 32$. In the upper part of the table, A and $-B$ have the eigenvalues $\lambda_i = -i, i = 1, 2, \ldots, m = n$, and $\alpha = \beta = 1.0$. For the lower part A and $-B$ have the eigenvalues: a) $\lambda_{Ai} = -i, \lambda_{Bi} = -1000.0 \cdot i^{-1}$, $\alpha = \beta = 1.0$, b) $\lambda_{Ai} = -i, \lambda_{Bi} = -2000.0 \cdot i^{-1}, \alpha = \beta = 1.0$, c) A and $-B$ have the eigenvalues $\lambda_i = -i, \alpha_A = \alpha_B = \beta_A = 1.0, \beta_B = 50.0$

			`PGESYCTD`			`psb04md`			Performance ratios		
$m = n$	sep^{-1}	$P_r \times P_c$	T_p	$\Vert X - \tilde{X}\Vert_F$	$\Vert R\Vert_F$	T_p	$\Vert X - \tilde{X}\Vert_F$	$\Vert R\Vert_F$	q_T	q_X	q_R
1024	2.5E-3	1×1	277	6.4E-12	3.5E-9	101	2.9E-12	2.4E-9	2.74	2.21	1.46
1024	2.4E-3	2×2	136	6.6E-12	3.5E-9	44	2.8E-12	2.3E-9	3.09	2.36	1.52
1024	2.4E-3	3×3	54	6.2E-12	3.5E-9	27	2.8E-12	2.3E-9	2.00	2.21	1.52
1024	2.3E-3	4×4	50	6.4E-12	3.6E-9	21	2.8E-12	2.3E-9	2.38	2.29	1.57
1024	2.3E-3	5×5	41	7.1E-12	3.6E-9	18	2.8E-12	2.3E-9	2.28	2.54	1.57
1024	2.3E-3	6×6	30	6.2E-12	3.6E-9	16	2.8E-12	2.3E-9	1.88	2.21	1.57
1024	2.3E-3	7×7	29	7.0E-12	3.6E-9	14	2.8E-9	2.3E-9	2.07	2.50	1.57
1024	2.5E-3	8×8	27	5.9E-12	3.4E-9	18	2.9E-12	2.4E-9	1.50	2.03	1.17
2048	1.3E-3	1×1	2053	2.1E-11	1.9E-8	1166	1.0E-11	1.8E-8	1.76	2.10	1.06
2048	1.2E-3	2×2	763	1.8E-11	1.9E-8	269	1.0E-11	1.7E-8	2.84	1.80	1.12
2048	1.2E-3	3×3	505	1.9E-11	2.0E-8	211	1.0E-11	1.7E-8	2.50	1.90	1.18
2048	1.2E-3	4×4	331	1.9E-11	1.9E-8	120	1.0E-11	1.7E-8	2.76	1.90	1.12
2048	1.2E-3	5×5	181	1.9E-11	2.0E-8	87	1.0E-11	1.7E-8	2.08	1.90	1.18
2048	1.2E-3	6×6	143	1.9E-11	2.0E-8	71	1.0E-11	1.7E-8	2.01	1.90	1.18
2048	1.2E-3	7×7	128	1.9E-11	2.0E-8	63	1.0E-11	1.7E-8	2.03	1.90	1.18
2048	1.3E-3	8×8	140	2.0E-11	2.0E-8	54	1.0E-11	1.7E-8	2.59	2.00	1.18
4096	7.0E-4	2×2	6514	5.2E-11	1.1E-7	3174	3.8E-11	1.3E-7	2.05	1.37	0.85
4096	7.0E-4	3×3	3338	5.0E-11	1.1E-7	1131	3.8E-11	1.3E-7	2.95	1.32	0.85
4096	7.0E-4	4×4	2912	5.0E-11	1.1E-7	909	3.8E-11	1.3E-7	3.20	1.32	0.85
4096	7.0E-4	5×5	1466	5.1E-11	1.1E-7	526	3.8E-11	1.3E-7	2.79	1.34	0.85
4096	7.0E-4	6×6	1027	5.1E-11	1.1E-7	398	3.8E-11	1.3E-7	2.58	1.34	0.85
4096	7.0E-4	7×7	804	5.8E-11	1.1E-7	342	3.8E-11	1.3E-7	2.35	1.53	0.85
4096	6.6E-4	8×8	890	5.4E-11	1.1E-7	295	3.8E-11	1.3E-7	3.02	1.42	0.85
8192	mem	4×4	15543	1.6E-10	7.5E-7	7425	1.5E-10	9.8E-7	2.09	1.07	0.77
8192	3.5E-4	5×5	10435	1.5E-10	7.2E-7	3817	1.5E-10	9.9E-7	2.73	1.00	0.73
8192	3.5E-4	6×6	7987	1.5E-10	6.4E-7	2740	1.5E-10	9.9E-7	2.91	1.00	0.65
8192	3.5E-4	7×7	6224	1.7E-10	6.5E-7	2197	1.5E-10	9.8E-7	2.83	1.13	0.66
8192	3.6E-4	8×8	5313	1.7E-10	6.8E-7	2247	1.5E-10	9.9E-7	2.36	1.13	0.69
1024^a	9.2	1×1	274	4.3E-9	2.2E-9	73	2.1E-5	1.2E-2	3.75	2.0E-4	1.8E-7
1024^a	8.2	2×2	121	6.3E-9	2.1E-9	36	1.2E-5	7.2E-3	3.36	5.3E-4	2.9E-7
1024^a	5.9	3×3	52	4.7E-9	2.1E-9	22	2.1E-5	1.2E-2	2.36	2.2E-4	1.8E-7
1024^a	7.2	4×4	51	3.3E-9	2.1E-9	18	3.1E-5	1.8E-2	2.83	1.1E-4	1.2E-7
1024^a	4.9	5×5	35	5.2E-9	2.1E-9	15	3.5E-5	2.1E-2	2.33	1.5E-4	1.0E-7
1024^a	6.4	6×6	28	5.5E-9	2.1E-9	12	1.9E-5	1.1E-2	2.33	2.9E-4	1.2E-7
1024^a	9.0	7×7	28	6.3E-9	2.1E-9	12	1.9E-5	1.2E-2	2.33	3.3E-4	1.8E-7
1024^a	4.3	8×8	26	4.2E-9	2.1E-9	10	1.6E-5	9.6E-3	2.60	2.6E-4	2.2E-7
2048^b	5.1E4	1×1	2223	2.2E-5	2.3E-8	1211	0.34	389	1.84	6.5E-4	5.9E-11
2048^b	4.9E4	2×2	859	1.7E-5	2.3E-8	284	0.36	407	3.02	4.7E-4	5.7E-11
2048^b	6.4E4	3×3	496	3.4E-5	2.3E-8	178	0.41	474	2.79	8.3E-5	4.9E-11
2048^b	5.1E4	4×4	356	6.3E-5	3.1E-8	132	0.36	421	2.70	1.8E-4	7.4E-11
2048^b	6.6E4	5×5	189	4.4E-5	2.3E-8	95	0.36	428	1.99	1.2E-4	5.4E-11
2048^b	6.7E4	6×6	152	8.5E-5	2.3E-8	77	0.34	422	1.97	2.5E-4	5.5E-11
2048^b	6.7E4	7×7	127	3.5E-5	2.3E-8	68	0.36	412	1.87	9.7E-5	5.6E-11
2048^b	6.6E4	8×8	145	3.1E-5	2.3E-8	58	0.35	394	2.50	8.9E-5	5.8E-11
4096^b	1.3E6	2×2	6470	8.1E-3	1.2E-7	24008	dnc	dnc	0.27	0.0	0.0
4096^b	6.6E4	3×3	3611	2.2E-4	1.2E-7	1470	63.3	1.5E5	2.46	3.5E-6	8.0E-13
4096^b	5.6E4	4×4	2271	1.9E-4	5.1E-6	1202	57.9	1.4E5	1.89	3.3E-6	3.6E-11
4096^b	5.8E4	5×5	1628	2.8E-4	1.2E-7	4201	dnc	dnc	0.39	0.0	0.0
4096^b	6.4E4	6×6	1134	2.3E-4	1.2E-7	476	62.4	1.5E5	2.38	3.7E-6	8.0E-13
4096^b	8.4E4	7×7	839	2.2E-4	1.2E-5	409	44.9	1.0E5	2.05	4.9E-6	1.2E-10
4096^b	6.7E4	8×8	916	1.5E-4	1.2E-7	337	51.5	1.2E5	2.72	2.9E-6	1.0E-12
8192^b	mem	4×4	16538	5.6E-4	6.4E-7	7652	1.22	5.7E3	2.16	4.6E-4	1.1E-10
8192^b	2.5E4	5×5	10532	7.5E-4	6.6E-7	4037	1.22	5.7E3	2.61	6.1E-4	1.2E-10
8192^b	2.5E4	6×6	8388	2.7E-4	6.3E-7	3146	0.87	4.1E3	2.67	3.1E-4	1.5E-10
8192^b	4.8E4	7×7	5623	3.5E-4	6.3E-7	2387	0.37	1.8E3	2.36	9.5E-4	3.5E-10
8192^b	3.4E4	8×8	5365	8.8E-4	6.0E-7	2240	0.67	2.4E3	2.40	1.3E-4	2.5E-10

and Z_{SYCT} is the $mn \times mn$ matrix representation of the Sylvester operator defined by $Z_{\text{SYCT}} = I_n \otimes A + B^T \otimes I_m$. Moreover, $\text{sep}(A, B)$ is a condition number for the SYCT equation, but it is expensive to compute in practice ($O(m^3 n^3)$ operations). However, we can compute a lower bound of $\text{sep}(A, B)^{-1} = \Vert Z_{SYCT}^{-1}\Vert_2$ in parallel, which is based on the same technique as described in [13,11]. Since a tiny value of $\text{sep}(A, B)$ signals ill-conditioning for SYCT, the sep^{-1}-estimate signals ill-conditioning when it

Table 3. Speed and accuracy results on Super Linux Cluster `seth` for the routines `PGESYCTD` and `psb04md` solving the triangular ($Q_A = Q_B = I$) equation $AX - XB = C$, A and B in real Schur form for well-conditioned problems. All problems use the blocking factors *mb* $=$ *nb* $= 128$. A and $-B$ have the eigenvalues $\lambda_i = -i, i = 1, 2, \ldots, m = n$, and $\alpha = \beta = 1.0$

			`PGESYCTD`			`psb04md`			Performance ratios		
$m = n$	sep^{-1}	$P_r \times P_c$	T_p	$\Vert X - \tilde{X}\Vert_F$	$\Vert R\Vert_F$	T_p	$\Vert X - \tilde{X}\Vert_F$	$\Vert R\Vert_F$	q_T	q_X	q_R
1024	2.5E-3	1×1	2.6	3.2E-13	3.8E-10	94.3	3.8E-13	4.1E-10	2.7E-2	0.84	0.93
1024	2.5E-3	2×2	1.7	3.9E-13	5.3E-10	45.7	4.7E-13	5.5E-10	3.7E-2	0.83	0.96
1024	2.5E-3	3×3	1.6	3.9E-13	4.9E-10	30.3	4.7E-13	5.1E-10	5.2E-2	0.83	0.96
1024	2.5E-3	4×4	1.5	3.8E-13	5.2E-10	23.5	4.7E-13	5.5E-10	6.4E-2	0.81	0.95
1024	2.5E-3	5×5	1.4	4.0E-13	4.7E-10	22.9	4.8E-13	5.0E-10	6.1E-2	0.83	0.94
1024	2.5E-3	6×6	1.4	4.1E-13	4.7E-10	22.5	4.9E-13	5.0E-10	6.2E-2	0.84	0.94
1024	2.5E-3	7×7	1.4	4.4E-13	5.1E-10	21.1	5.0E-13	5.3E-10	6.6E-2	0.88	0.96
1024	$1.1E-3$	8×8	1.0	3.9E-13	5.5E-10	17.0	4.5E-13	5.8E-10	5.9E-2	0.87	0.95
2048	1.3E-3	1×1	17.8	1.2E-12	3.1E-9	1116	1.1E-12	2.3E-9	1.6E-2	1.01	1.35
2048	1.3E-3	2×2	9.5	1.6E-12	4.2E-9	287	1.5E-12	3.7E-9	3.3E-2	1.07	1.14
2048	1.3E-3	3×3	7.9	1.6E-12	3.9E-9	166	1.5E-12	3.4E-9	4.8E-2	1.07	1.15
2048	1.3E-3	4×4	6.7	1.5E-12	4.0E-9	125	1.5E-12	3.5E-9	5.4E-2	1.00	1.14
2048	1.3E-3	5×5	5.9	1.6E-12	3.9E-9	114	1.6E-12	3.4E-9	5.2E-2	1.00	1.15
2048	1.3E-3	6×6	5.2	1.5E-12	3.8E-9	87.4	1.5E-12	3.2E-9	5.9E-2	1.00	1.19
2048	1.3E-3	7×7	5.0	1.6E-12	3.8E-9	83.8	1.6E-12	3.2E-9	6.0E-2	1.00	1.19
2048	1.3E-3	8×8	4.4	1.5E-12	4.1E-9	64.8	1.5E-12	3.6E-9	6.7E-2	1.00	1.14
4096	7.0E-4	1×1	129	4.9E-12	2.5E-8	8812	3.0E-12	1.3E-8	1.5E-2	1.63	1.92
4096	6.9E-4	2×2	63.4	6.1E-12	3.3E-8	2687	5.1E-12	2.6E-8	2.3E-2	1.20	1.27
4096	6.9E-4	3×3	45.7	6.1E-12	3.2E-8	1018	5.4E-12	2.4E-8	4.5E-2	1.13	1.33
4096	6.9E-4	4×4	37.1	6.2E-12	3.2E-8	701	5.3E-12	2.4E-8	5.3E-2	1.17	1.33
4096	6.9E-4	5×5	29.3	6.3E-12	3.1E-8	554	5.5E-12	2.3E-8	5.3E-2	1.15	1.35
4096	6.9E-4	6×6	25.3	6.3E-12	3.0E-8	439	5.6E-12	2.3E-8	5.8E-2	1.13	1.30
4096	6.9E-4	7×7	25.6	6.2E-12	3.0E-8	385	5.6E-12	2.2E-8	6.6E-2	1.10	1.36
4096	6.9E-4	8×8	22.3	6.2E-12	3.0E-8	326	5.5E-12	2.4E-8	6.8E-2	1.13	1.25
8192	3.5E-4	4×4	235	2.5E-11	2.5E-7	6003	2.0E-11	1.8E-7	3.9E-2	1.25	1.39
8192	3.5E-4	5×5	167	2.5E-11	2.5E-7	dnc	dnc	dnc	0.0	0.0	0.0
8192	3.5E-4	6×6	139	2.6E-11	3.5E-7	2202	2.1E-11	1.6E-7	6.3E-2	1.23	2.19
8192	3.5E-4	7×7	146	2.5E-11	2.5E-7	2045	2.1E-11	1.7E-7	7.1E-2	1.19	1.47
8192	3.5E-4	8×8	128	2.6E-11	2.5E-7	dnc	dnc	dnc	0.0	0.0	0.0

gets large. We expect the explicitly blocked method to handle ill-conditioned problems better than the fully iterative, since it relies on orthogonal transformations and it has a direct (backward stable) solution method for the reduced triangular problem. To signal when there is not enough memory for both solving the problem and doing condition estimation, we use the notation mem.

5 Discussion of Computational Results

The experimental results from the last section reveal the following: For triangular problems ($Q_A = Q_B = I$), see Table 3 which displays results from the Linux Cluster, the parallel Bartels-Stewart based method is very much faster than the fully iterative since it always computes the solution directly using a fixed number of arithmetic operations.

For general problems ($Q_A \neq I, Q_B \neq I$) the results differ depending on the target machine. For the IBM SP system (see Table 1), `PGESYCTD` is able to compete with the fully iterative method regarding both speed, accuracy and residual errors, for both well- and ill-conditioned problems. For example, `PGESYCTD` uses only 33% (well-conditioned case) and 25% (ill-conditioned case) of the execution time of `psb04md` for $m = n = 4096, P_r = 4, P_c = 4$. For the ill-conditioned problems, the difference

Table 4. Hardware characteristics for Super Linux Cluster and IBM SP System. The parameter t_a denotes the time for performing an arithmetic operation, t_s denotes the experimentally measured startup time for message passing, t_n denotes the time to transfer one byte over a single link in the network and t_m denotes the peak time to transfer one byte through the memory of a node. The SP system has 3 times better flop/network bandwidth ratio and over 12 times better flop/memory bandwidth ratio than the Super Linux Cluster

Hardware	Super Linux Cluster	IBM SP System	Parameter	Super Linux Cluster	IBM SP System
CPU	120×2 Athlon MP2k+	64 thin P2SC	t_a	3.0×10^{-10} sec.	2.1×10^{-9} sec.
&	1.667Ghz nodes,	120 MHz nodes,	t_s	3.7×10^{-6} sec.	4.0×10^{-5} sec.
Memory	1-4 Gb/node,	128 Mb/node,	t_n	3.0×10^{-9} sec.	6.7×10^{-9} sec.
	peak 800 Gflops/sec.	peak 33.6 Gflops/sec.	t_m	9.6×10^{-10} sec.	5.6×10^{-10} sec.
Network	Wolfkit3 SCI h s i,	Multistage network,	t_a/t_s	8.1×10^{-5}	5.3×10^{-5}
	3-dim. torus,	peak 150 Mb/sec.	t_a/t_n	0.10	0.31
	peak 667 Mb/sec.		t_a/t_m	0.31	3.8

in accuracy on the IBM SP system is remarkable: as expected, the explicitly blocked method gives far more accuracy than the fully iterative, which sometimes did not even converge. For the Linux-Cluster (see Table 2), `psb04md` is about two or three times faster than `PGESYCTD` for both types of problems. This difference in speed can be explained by the different characteristics of the machines, see Table 4. For uniprocessor runs `psb04md` is faster on both machines, but when we go in parallel the superior memory and network bandwidth of the SP system makes it possible for `PGESYCTD` to scale much better than `psb04md`. On the less balanced Super Linux Cluster the heavy communication in `PGESYCTD` becomes a bottleneck.

For the ill-conditioned problems `PGESYCTD` gives the best accuracy, both regarding the forward error $\|X - \widetilde{X}\|_F$ and the residual error $\|R\|_F$ up to magnitudes 6 and 12, respectively. For the well-conditioned problems, the routines in general have forward and residual errors of the same order, even if `psb04md` shows slightly better forward error (up to 63% for the largest problems on the Linux Cluster ($m = n = 4096, 8192$)), and `PGESYCTD` shows slightly better residual error (up to 35% for the largest problems on the Linux Cluster). We remark that the column sep^{-1} in Tables 1–3 are lower bounds on the exact values.

6 Summary and Future Work

We have presented a comparison of parallel ScaLAPACK-style implementations of two different methods for solving the continuous-time Sylvester equation (1), the Bartels-Stewart method and the iterative matrix-sign-function-based method. The comparison has dealt with generality, speed and accuracy.

Experiments carried out on two different distributed memory machines show that the parallel explicitly blocked Bartels-Stewart algorithm can solve more general problems and delivers far more accuracy for ill-conditioned problems. A method that imposes more restrictions on the spectra on A and B in $AX - XB = C$ is considered less general. Ill-conditioning is measured with respect to the separation of A and B, $\mathrm{sep}(A, B)$, as defined in equation (8). We remark that $\mathrm{sep}(A, B)$ can be much smaller than the minimum distance between the eigenvalues of A and B. This means that we can have an ill-conditioned problem even if the spectra of A and B are well-separated, which for some examples could favour the iterative matrix-sign-function-based method. The Bartels-Stewart method is also up to four times faster for large enough problems on the most balanced parallel platform (IBM SP), while the parallel iterative algorithm is almost always the fastest of the two on the less balanced platform (HPC2N Linux Super Cluster).

Ongoing work includes implementing general Bartels–Stewart solvers for related matrix equations, e.g., the continuous-time Lyapunov equation $AX + XA^T = C$, $C = C^T$ and the discrete-time Sylvester equation $AXB^T - X = C$. Our objective is to construct a software package $SCASY$ of ScaLAPACK-style algorithms for the most common matrix equations, including generalized forms of the Sylvester/Lyapunov equations.

Acknowledgements

This research was conducted using the resources of the High Performance Computing Center North (HPC2N). Financial support has been provided by the *Swedish Research Council* under grant VR 621-2001-3284 and by the *Swedish Foundation for Strategic Research* under grant A3 02:128.

References

1. R.H. Bartels and G.W. Stewart Algorithm 432: Solution of the Equation $AX + XB = C$, *Comm. ACM*, 15(9):820–826, 1972.
2. E. Anderson, Z. Bai, C. Bischof. J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenny, S. Ostrouchov and D. Sorensen. *LAPACK User's Guide*. Third Edition. SIAM Publications, 1999.
3. P. Benner, E.S. Quintana-Orti. Solving Stable Generalized Lyapunov Equations with the matrix sign functions, *Numerical Algorithms*, 20 (1), pp. 75-100, 1999.
4. P. Benner, E.S. Quitana-Orti, G. Quintana-Orti. Numerical Solution of Discrete Schur Stable Linear Matrix Equations on Multicomputers, *Parallel Alg. Appl.*, Vol. 17, No. 1, pp. 127-146, 2002.
5. P. Benner, E.S. Quitana-Orti, G. Quintana-Orti. Solving Stable Sylvester Equations via Rational Iterative Schemes, Preprint SFB393/04-08, TU Chemnitz, 2004.
6. S. Blackford, J. Choi, A. Clearly, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide*. SIAM Publications, Philadelphia, 1997.
7. R. Granat, A Parallel ScaLAPACK-style Sylvester Solver, *Master Thesis*, UMNAD 435/03, Dept. Computing Science, Umeå University, Sweden, January, 2003.
8. R. Granat, B. Kågström, P. Poromaa. Parallel ScaLAPACK-style Algorithms for Solving Continous-Time Sylvester Equations. In H. Kosch et al., Euro-Par 2003 Parallel Processing. *Lecture Notes in Computer Science*, Vol. 2790, pp. 800-809, 2003.
9. G. Henry and R. Van de Geijn. Parallelizing the QR Algorithm for the Unsymmetric Algebraic Eigenvalue Problem: Myths and Reality. *SIAM J. Sci. Comput.* 17:870–883, 1997.
10. G. Henry, D. Watkins, and J. Dongarra. A Parallel Implementation of the Nonsymmetric QR Algorithm for Distributed Memory Architectures. Technical Report CS-97-352 and Lapack Working Note 121, University of Tennessee, 1997.

11. N.J. Higham. Perturbation Theory and Backward Error for $AX - XB = C$, *BIT*, 33:124–136, 1993.
12. B. Kågström, P. Ling, and C. Van Loan. GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Software*, 24(3):268–302, 1998.
13. B. Kågström and P. Poromaa. Distributed and shared memory block algorithms for the triangular Sylvester equation with Sep^{-1} estimators, *SIAM J. Matrix Anal. Appl.*, 13 (1992), pp. 99–101.
14. Niconet Task II: Model Reduction, website: www.win.tue.nl/niconet/NIC2/NICtask2.html
15. P. Poromaa. Parallel Algorithms for Triangular Sylvester Equations: Design, Scheduling and Scalability Issues. In Kågström et al. (eds), *Applied Parallel Computing. Large Scale Scientific and Industrial Problems*, Lecture Notes in Computer Science, Vol. 1541, pp 438–446, Springer-Verlag, 1998.
16. J.D. Roberts. Linear model reduction and solution of the algebraic Ricatti equation by use of the sign function, *Intern. J. Control*, 32:677-687, 1980. (Reprint of Technical Report No. TR-13, CUED/B-Control, Cambridge University, Engineering Department, 1971).
17. SLICOT library in the Numerics in Control Network (NICONET), website: www.win.tue.nl/niconet/index.html

Performance Analysis for Parallel Adaptive FEM on SMP Clusters

Judith Hippold[⋆] and Gudula Rünger

Chemnitz University of Technology, Department of Computer Science
09107 Chemnitz, Germany
{juh,ruenger}@informatik.tu-chemnitz.de

Abstract. The parallel implementation of 3-dimensional adaptive finite element methods (FEM) on recent clusters of SMPs has to deal with dynamic load increase due to adaptive refinement which might result in load imbalances. Additionally, the irregular behavior of adaptive FEM causes further impacts like varying cache effects or communication costs which make a redistribution strategy more complex. In this paper we investigate the interaction of these effects and the performance behavior of parallel adaptive FEM and present several performance results on recent SMP clusters.

1 Introduction

Finite element (FE) methods are frequently used to solve problems in natural sciences and engineering modeled by partial differential equations (PDE). FE methods discretize the physical domain into a mesh of finite elements and approximate the unknown solution function by a set of shape functions on those elements. Adaptive mesh refinement (AMR) has been designed to reduce computational effort and memory needs. But, especially for simulations of real-life problems with 3-dimensional, hexahedral meshes parallel execution is still necessary to provide a satisfying solution in reasonable time. A parallel execution of adaptive FEM on distributed memory machines, however, usually exhibits load imbalances resulting from adaptive refinements concentrated on a few processors. Thus load redistribution is necessary but causes additional overhead and a tradeoff between redistribution costs and load balance has to be found in order to allow an efficient dynamic redistribution strategy. The use of a specific parallel platform further influences the parallel execution time. In this paper we consider clusters of SMPs.

Clusters of SMPs (symmetric multiprocessors) gain more and more importance in high performance computing because they offer large computing power for a reasonable price. We have realized a parallel program for adaptive FEM [2] working on adaptively refined, hexahedral meshes and have investigated the execution on SMP cluster architectures. First runtime experiments with parallel adaptive hexahedral FEM on SMP clusters have shown a quite irregular behavior of parallel runtime and speedup values. This includes constant speedup behavior despite load increase due to refinement, superlinear speedups also for unbalanced load, or saturation of speedups for quite small numbers of processors. These effects vary for different application problems and are

⋆ Supported by DFG, SFB393 *Numerical Simulation on Massively Parallel Computers*

further influenced when using SMP clusters and exploiting them differently for communication. Compared to single-node clusters the two-level architecture of SMP clusters offers different communication realizations, where SMP internal communication might be implemented more efficiently. Well-known redistribution strategies cannot capture this irregular behavior. The goal of this paper is to investigate the irregular effects of performance behavior caused by adaptive parallel FEM on SMP clusters and to analyze reasons for these irregular effects, which is a necessary prerequisite for dynamic redistribution.

Possible reasons for irregular parallel execution times or speedups of adaptive methods are cache effects, non-uniform communication time on SMP clusters, and varying load increase due to mesh refinement. In this paper we present runtime experiments for the parallel adaptive FEM implementation on SMP clusters for different aspects of load increase, load imbalances, and architecture specific properties. Especially we consider cache effects for basic matrix operations intensively used in the program. Also the communication behavior on SMP clusters is investigated. Results are presented for three different cluster platforms, which show a strong influence of the specific architecture. The runtime experiments exhibit quite diverse effects. The contribution of this paper is to gather important influential reasons for an irregular runtime behavior. To understand these reasons is important for a potential adaptive load balancing strategy on SMP clusters.

The paper is structured as follows: Section 2 introduces the parallel adaptive FEM implementation. Section 3 illustrates irregular runtime and speedup effects on SMP clusters. Section 4 presents several specific runtime experiments and Section 5 concludes.

2 Parallel Adaptive FEM Implementation

The parallel adaptive FEM implementation [2] is based on a sequential version [1] and is used to solve systems of 2nd order partial differential equations, such as the Poisson equation (1) or the Lamé system of linear elasticity (2) (see [1]).

$$Lu := -\nabla \cdot (A(x)\nabla u) + cu = f \ \ in\ \Omega \subset I\!R^3, \qquad A(x) = diag(a_i)_{i=1}^3 \tag{1}$$
$$u = u_0 \quad on\ \partial\Omega_1, \qquad n^t A(x)\nabla u = g \quad on\ \partial\Omega_2$$

$$-\mu\Delta\underline{u} - (\lambda+\mu)\, grad\, div\, \underline{u} = \underline{f} \quad in\ \Omega \subset I\!R^3, \qquad \underline{u} = (u^{(1)}, u^{(2)}, u^{(3)})^t \tag{2}$$
$$u^{(i)} = u_0^{(i)} \quad on\ \partial\Omega_1^{(i)}, \qquad t^{(i)} = g^{(i)} \quad on\ \partial\Omega_2^{(i)}, \quad i = 1,2,3$$

In the parallel version the finite elements are spread over the available processors and the global stiffness matrix is represented in local memories by triangular element stiffness matrices. Nodes of FEs shared between processors are duplicated and exist in the address space of each owning processor. The different owners of duplicates calculate only subtotals which have to be accumulated using communication operations to yield the totals. For the parallel implementation a specific numerical parallelization approach is adopted from [6,7,8] so that the processors mainly work on unaccumulated vectors. The design of the parallel adaptive FEM implementation reduces the number of data exchanges by separating computation phases from communication phases [3] (see also Section 4.2). The following pseudo-code illustrates the coarse structure of the implementation:

```
create initial mesh;
while(convergence is not achieved) {
*** Refinement phase (REF) ***
        while((there are FEs labeled by the error estimator)    OR
                (neighboring FEs have refinement level difference larger than 1))
                        refine these FEs;
*** Repartitioning phase ***
        calculate a new partitioning and redistribute;
*** Assembling phase (ASS) ***
        foreach(new FE) assemble the element stiffness matrix;
*** Solution phase (SOL) ***
        while(convergence is not achieved)
                        solve the system of equations with the preconditioned
                        conjugate gradient method;
*** Error estimation phase (EST) ***
        estimate the error per FE;
}
```

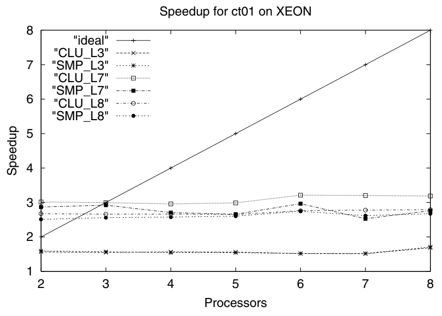
The program executes the five phases within the outer while-loop until the maximum error drops below a predefined threshold value. The outer loop includes two iterative phases: adaptive mesh refinement (REF) and solving the system of equations (SOL). REF is performed until a valid mesh is created which means all FEs labeled for refinement are subdivided and the difference of the refinement level between all neighboring elements is less than two. SOL uses the iterative preconditioned conjugate gradient method to solve the system of equations [6].

3 Parallel Runtime Behavior

This section investigates the parallel runtime behavior of the adaptive FEM implementation on SMP clusters. In order to capture the different effects, we perform several runtime experiments with varying load, varying load imbalances, and different machine utilization. As hardware platforms the three following machines are used:

XEON:	a Xeon cluster with 16 nodes and two 2.0 GHz Intel Xeon processors per node (SCI, ScaMPI [4], 512 KB L2 cache),
SB1000:	a dual cluster of four Sun Blade 1000 with 750 MHz Ultra Sparc3 processors (SCI, ScaMPI [4], 8 MB L2 cache), and
JUMP:	an IBM Regatta system with 41 nodes and 32 1.7 GHz Power4+ processors per node (High Performance Switch, Parallel Environment, 1.5 MB L2 cache per 2 processors, 512 MB L3 cache per 32 processors).

Figure 1 shows speedups on XEON for ct01, solving the Lamé equation ($\Omega = (0,2) \times (0,1) \times (0,4)$; $\partial\Omega_1^{(i)} = (0,1) \times (0,1) \times \{0\}$, $i = 1,2,3$; $\partial\Omega_2^{(3)} = (0,2) \times$



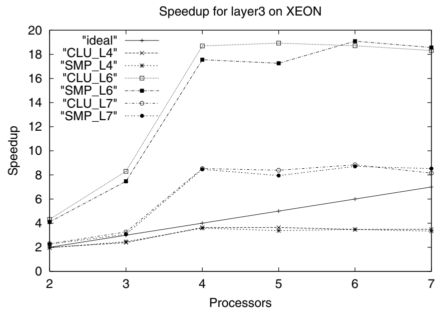


Fig. 1. Speedups for examples ct01 (left) and layer3 (right) on XEON. L denotes the different refinement levels measured. CLU and SMP distinguish the assignment of MPI processes to processors. The CLU version runs only one process per cluster node and the SMP version assigns MPI processes to processors of the same cluster node rather than to different nodes

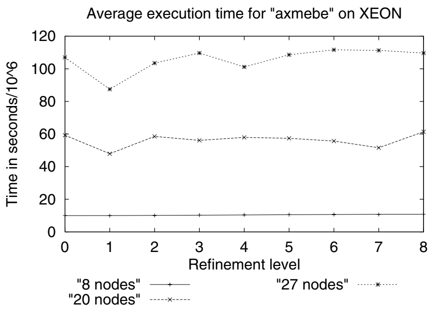
$(0, 1) \times \{4\}$; $g^{(3)} = 1000$ (see [1])), and layer3, a boundary layer for the convection-diffusion equation ($-\epsilon\Delta u + u = 1 \quad in\ \Omega = (0, 1)^3; \quad u = 0\ for\ x, y, z = 0; \quad \frac{\partial u}{\partial n} = 0\ for\ x, y, z = 1$ (see [1])) . The execution times are measured for different numbers of adaptive refinement steps and program runs without repartitioning. We distinguish the CLU and SMP process assignment strategies. CLU assigns MPI processes only to different cluster nodes and SMP assigns MPI processes to processors of the same cluster node rather than to different nodes.

For the example application ct01 the load concentrates on two processors. Thus increasing the number of processors does not have effects on the parallel runtime. With growing refinement level (L7, L8) the load for the two mainly employed processors is increasing and slightly superlinear speedups can be observed. For this levels also a significant difference between the different process assignment strategies emerges: the CLU strategy achieves better results.

For the example application layer3 the imbalance of load is not as heavy as for ct01. Even for small refinement levels (L4) good speedups can be measured up to four processors. Larger meshes (L6, L7) create strongly superlinear speedups which remain constant or drop only slightly when the number of processors is increasing and is larger than four although the load for the processors develops unevenly. The CLU process assignment strategy achieves in most cases higher speedups. These example applications illustrate the irregular behavior of parallel runtime and speedup values for adaptive FEM on SMP clusters.

4 Determining and Quantifying the Dependences

This section examines the impact of different hardware characteristics and algorithmic properties on the execution time of the adaptive FEM implementation in order to determine reasons for the runtime behavior illustrated in Section 3. We present measurements for a program solving the Lamé equation on the input meshes called M8, M64, and M512 which mainly differ in the number of initial volumes.



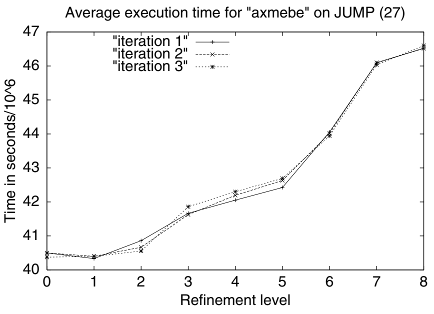


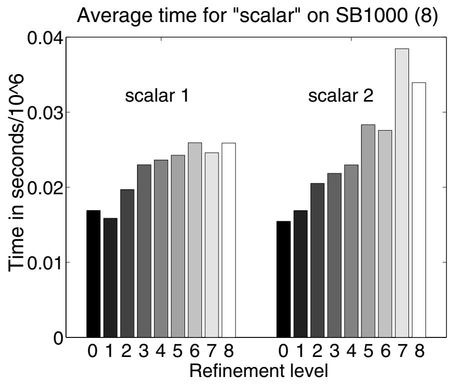
Fig. 2. Left: Average execution times for the operation axmebe on XEON for different finite elements (8, 20, 27 nodes per FE) and different refinement levels. Right: Average execution times for distinct solver iterations (first, second, third iteration) on JUMP (27 nodes per FE)

4.1 Cache Behavior

When the number of finite elements grows the number of different memory accesses is increasing and cache misses can influence performance disadvantageously. Thus superlinear speedups are possible for parallel program runs on reduced data sets. To investigate the cache behavior the sequential execution of operations working on large data structures is regarded on different refinement levels. These operations are axmebe, axpy, and scalar of phase SOL.

Axmebe: The operation axmebe performs the matrix-vector multiplication $\underline{y} = \underline{y} + A * \underline{u}$ where A is the triangular element stiffness matrix and $\underline{y}$ and $\underline{u}$ are global vectors. During program execution the size of A is constant and is between 2.3 KB and 26 KB depending on the finite element type. The length of the global vectors $\underline{y}$ and $\underline{u}$ varies according to the current refinement level of the mesh and additionally depends on the FE type. The memory requirements for the vectors in the program run used for Figure 2 is about 6.3 KB at refinement level three and 70.7 KB at refinement level eight for FEs with 8 nodes. For elements with 27 nodes the requirement is 36.6 KB and 453.4 KB at the corresponding levels. The global vectors are accessed irregularly because the vector entries for the nodes of a finite element are not stored consecutively. This is due to the shared nodes between the different elements. Figure 2 (left) illustrates the average execution times for the operation axmebe running on finite elements with 8, 20, and 27 nodes on one processor of XEON. The execution times vary for finite elements creating larger vectors and matrices. Although the average execution time for one vector entry over the entire number of iterations necessary for phase SOL is calculated, drops in the average execution time emerge for small numbers of solver iterations. This behavior can be observed for all platforms investigated. Thus we have additionally measured distinct solver iterations to reduce the effects of e. g. context switches: On the right of Figure 2 the average execution times for the first, second, and third solver iteration for a program run on JUMP with 27 nodes per FE are shown.

Scalar: The operation scalar calculates the scalar product $prod = \underline{x} * \underline{y}$ of the global vectors $\underline{x}$ and $\underline{y}$. The average execution times to calculate one vector entry on SB1000



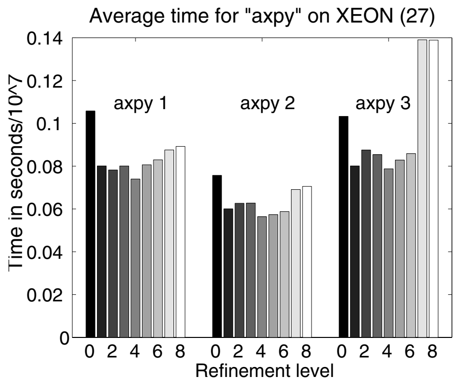


Fig. 3. Left: Average execution times for the operation scalar on different refinement levels (SB1000, 8 nodes per FE). Right: Average execution times for the operation axpy on different refinement levels (XEON, 27 nodes per FE)

are shown on the left of Figure 3 for different refinement levels. Scalar is performed two times during one solver iteration which is denoted by scalar 1 and scalar 2. The execution behavior of both scalar calls is similar but is influenced by the previously performed operation, especially for large vectors: If an input vector of the operation scalar has already been used as a parameter in the previous operation, scalar is faster. Analogous to the operation axmebe the number of iterations of phase SOL has influence on the average execution times.

Axpy: The average execution times to compute one vector entry for the operation axpy on XEON are shown on the right of Figure 3. Axpy calculates $\underline{y} = alpha * \underline{x} + \underline{y}$ where $\underline{y}$ and $\underline{x}$ are global vectors and $alpha$ is a scalar. As scalar, axpy is performed several times during one iteration of SOL and the average execution times of the different calls depend on the data used by the previous operation and the number of solver iterations.

Level	SB1000			XEON			JUMP		
	axmebe	scalar	axpy	axmebe	scalar	axpy	axmebe	scalar	axpy
1	30.05	0.36	0.63	30.71	1.18	1.90	76.25	6.60	9.09
3	78.49	0.79	1.23	34.11	0.78	0.47	77.27	2.43	2.83
5	80.61	0.69	1.00	37.73	0.30	0.52	80.18	1.16	1.48
7	81.33	0.67	1.00	37.84	0.42	0.27	74.51	0.87	1.28

Fig. 4. Percentage of time for the operations axmebe, scalar, and axpy on the total, sequential time of phase SOL using finite elements with 8 nodes

The three operations investigated show an increase in their average execution times for large finite element meshes and high numbers of FE nodes. Especially the operation axmebe constitutes a high proportion of the total solver runtime (see Table 4), so increases in execution time may have influence on the total runtime of the program.

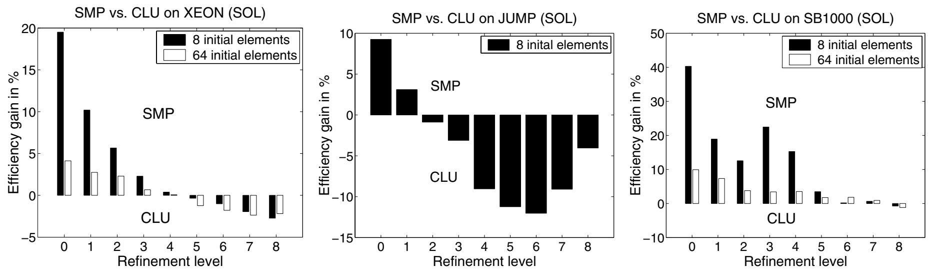


Fig. 5. Comparison of the process assignment strategies for phase SOL on the input meshes M8 and M64 on two processors of XEON, JUMP, and SB1000. Positive scale: efficiency gain of the SMP version (two MPI processes per cluster node); negative scale: efficiency gain of the CLU version (one MPI process per cluster node)

However, the increase on high refinement levels is small or the values even decrease on some platforms as illustrated in Table 4. Thus the impact is not strong enough to cause the heavy superlinear speedups shown in Section 3.

4.2 Communication Overhead

We consider the SMP and CLU process assignment strategies. If one MPI process is assigned to one processor, MPI processes situated at the same cluster node may take benefit from the internal SMP communication which is more efficient than network communication in many cases [5]. However, as shown in Section 3 the CLU assignment strategy can achieve better results than the SMP strategy, although the load is increasing. This subsection investigates the reasons by parallel program runs on two processors of different platforms. We use a maximum parallelism of two because there are only two CPUs per SMP on XEON and SB1000. Thus the independent consideration of both strategies is possible.

The parallel adaptive FEM implementation uses a special communication scheme. This scheme gathers the data to exchange and performs a global exchange (GE) after all local calculations are done. For the different algorithmic phases we have the following numbers of global exchanges:

Adaptive refinement (REF):	$refinement\ iterations * (2\,GE)$
Assemble element stiffness matrices (ASS):	$(1\,GE)$
Solve the system of equations (SOL):	$solver\ iterations * (6\,GE) + (2\,GE)$
Error estimation (EST):	$(2\,GE)$

The algorithmic design decouples the number of FEs and the number of necessary communication operations. Thus fast internal communication does not necessarily dominate network communication when the load is increasing.

Figure 5 compares the two strategies for the most communication-intensive phase SOL. On all platforms the SMP version is more efficient for low refinement levels. With increasing data (increased refinement level or increased size of the initial mesh) the benefit of the SMP strategy is lowered or the CLU version even dominates. Possible

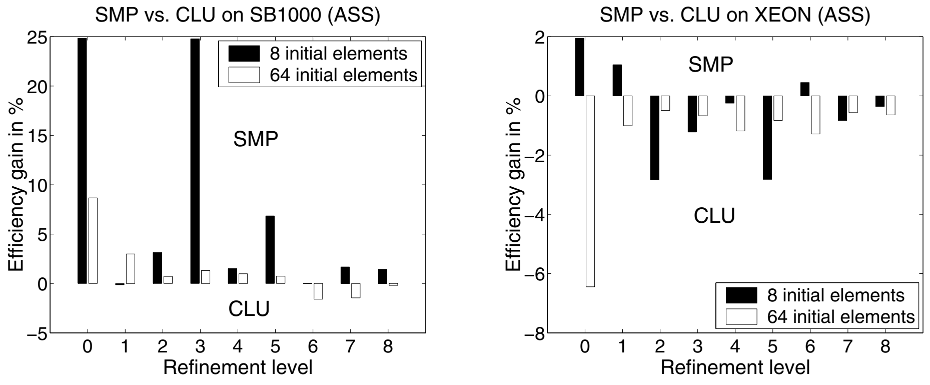


Fig. 6. Comparison of the process assignment strategies for assembling the element stiffness matrices on the input meshes M8 and M64 on two processors of SB1000 and XEON. Positive scale: efficiency gain of the SMP version (two MPI processes per cluster node), negative scale: efficiency gain of the CLU version (one MPI process per cluster node)

reasons are process scheduling or shared hardware components which cause additional overhead for the SMP version.

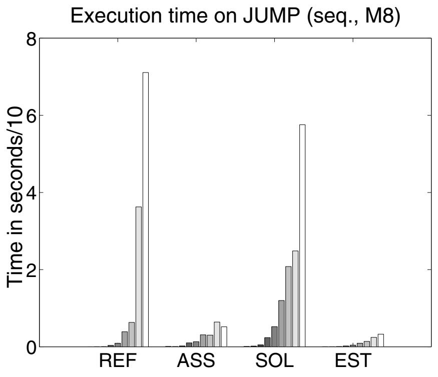
Figure 6 illustrates the impact of the process assignment strategy for the phase assembling the element stiffness matrices. During this phase only one global exchange is performed. Compared to Figure 5 the efficiency for the SMP version on SB1000 is lower. On XEON only for small numbers of finite elements the SMP version achieves better runtimes than the CLU assignment strategy. Due to the varying numbers of new finite elements on the different refinement levels there are strong fluctuations in efficiency.

The total execution time comprises runtimes of algorithmic phases with high and low communication needs and gathers the values of different refinement levels. So the benefits and drawbacks of the different strategies are combined. On XEON and SB1000 the differences in total runtime caused by the different process assignment strategies are approximately two percent for high refinement levels (levels 8 and 9) where on XEON the CLU version and on SB1000 the SMP version is slightly faster. On JUMP the SMP process assignment strategy is about 10 percent faster after 9 refinement steps for the investigated example. Thus a platform adapted assignment of MPI processes to processors can improve performance.

4.3 Algorithmic Properties

Subsection 4.2 has investigated the dependence between load increase and process assignment strategy and 4.1 has examined the dependence between load increase and cache behavior. However, the runtime behavior shown in Section 3, especially the superlinear speedup, cannot completely explained with these examinations. This subsection analyzes the influence of algorithmic properties.

Figure 7 shows execution times for the different algorithmic phases per different refinement level. On the left measurements for a sequential program run with mesh M8 on JUMP and on the right measurements for a parallel run on two processors with mesh



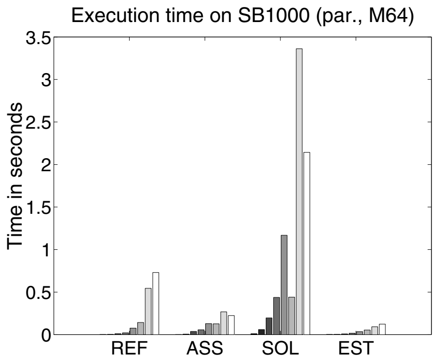


Fig. 7. Execution times per refinement level for the algorithmic phases REF, ASS, SOL, EST (left: sequential execution time on JUMP with mesh M8, right: parallel execution time on two processors of SB1000 with mesh M64)

M64 on SB1000 are shown. In general the diagrams show similar shapes for the different platforms. The most time consuming phases for parallel and sequential execution are the solution of the system of equations (SOL) and the adaptive mesh refinement (REF). The execution times increase with growing mesh size except for the phases ASS and SOL: For solving the system of equations this results from the different convergence behavior of the preconditioned conjugate gradient method and for assembling the element stiffness matrices this is caused by the unevenly growing number of new FEs and their element stiffness matrices which have to be created.

Especially on XEON and JUMP the phase REF shows very high sequential execution times compared to the parallel version. These originate from operations on lists of data structures with non-linear complexity. The length of these lists shrinks with growing parallelism. Thus superlinear speedups are possible if the load imbalances are not too strong. On SB1000 this effect has not been observed for the investigated example.

5 Conclusion and Future Work

In this paper we have presented several runtime experiments to analyze the irregular behavior of parallel execution and speedups of adaptive 3-dimensional FEM on recent SMP clusters. The investigations have shown that superlinear speedup behavior can occur for unbalanced load or saturation of speedup can occur for balanced load. Existing dynamic redistributions based on the mesh structure and load imbalances cannot capture these effects and are too expensive since a cost intensive redistribution might be applied although the efficiency of the program is already good. The reasons for this behavior are a conglomeration of only slight cache effects, of varying communication efficiency on SMP nodes, and efficiency changes in the algorithm, like a faster search in list structures. Thus the irregular effects cannot be predicted and for an efficient use of redistribution a decision based on explicit performance data seems reasonable. The advantage is that redistribution is only applied when necessary, and thus, redistributing is cheaper since the dynamic performance analysis is less expensive than redistribution operations. Our analysis also suggests to exploit SMP cluster nodes in a varying mode and to adapt the numbers of processors. Future work will realize the redistribution strategy based on a decision component that guides the redistribution according to the performance analysis.

References

1. S. Beuchler and A. Meyer. *SPC-PM3AdH v1.0, Programmer's Manual. Technical Report SFB393/01-08.* Chemnitz University of Technology, 2001.
2. J. Hippold, A. Meyer, and G. Rünger. An Adaptive, 3-Dimensional, Hexahedral Finite Element Implementation for Distributed Memory. In J. J. Dongarra, M. Bubak, G. D. van Albada, editor, *Proc. of Int. Conf. on Computational Science (ICCS04), LNCS 3037*, pages 149–157. Springer, Poland, Kraków, 2004.
3. J. Hippold and G. Rünger. A Data Management and Communication Layer for Adaptive, Hexahedral FEM. In M. Danelutto, D. Laforenza, M. Vanneschi, editor, *Proc. of the 10th Int. Euro-Par Conf., LNCS 3149*, pages 718–725 . Springer, Pisa, Italy, 2004.
4. http://www.scali.com
5. L. P. Huse, K. Omanga, H. Bugge, H. Ry, A. T. Haugsdal, and E. Rustad. ScaMPI - Design and Implementation.
6. A. Meyer. A parallel preconditioned conjugate gradient method using domain decomposition and inexact solvers on each subdomain. *Computing*, 45:217–234, 1990.
7. A. Meyer. Parallel Large Scale Finite Element Computations. In G. Cooperman, G. Michler, and H. Vinck, editors, *LNCIS 226*, pages 91–100. Springer, 1997.
8. A. Meyer and D. Michael. A modern approach to the solution of problems of classic elasto–plasticity on parallel computers. *Numerical Linear Algebra and Application*, 4(3):205–221, 1997.

Performance Tuning of Matrix Triple Products Based on Matrix Structure

Eun-Jin Im[1], Ismail Bustany[2], Cleve Ashcraft[3],
James W. Demmel[4], and Katherine A. Yelick[4]

[1] Kookmin University, Seoul, Korea
ejim@eecs.berkeley.edu
[2] Barcelona Design Inc., USA
isk@sierra-da.com
[3] Livermore Software Technology Corporation, USA
cleve@lstc.com
[4] U.C. Berkeley, USA
{demmel,yelick}@eecs.berkeley.edu

Abstract. Sparse matrix computations arise in many scientific and engineering applications, but their performance is limited by the growing gap between processor and memory speed. In this paper, we present a case study of an important sparse matrix triple product problem that commonly arises in primal-dual optimization method.

Instead of a generic two-phase algorithm, we devise and implement a single pass algorithm that exploits the block diagonal structure of the matrix. Our algorithm uses fewer floating point operations and roughly half the memory of the two-phase algorithm. The speed-up of the one-phase scheme over the two-phase scheme is 2.04 on a 900 MHz Intel Itanium-2, 1.63 on an 1 GHz Power-4, and 1.99 on a 900 MHz Sun Ultra-3.

1 Introduction

This paper contains a case study of an important matrix triple product problem that commonly arises in primal-dual optimization method [1]. The method is used in many practical applications such as circuit design, optimal control, portfolio optimization, etc., and the matrix triple product is often a bottleneck in solving the large-scale symmetric definite sparse matrix problems associated with primal-dual methods. The sparse matrix patterns are particular to the particular primal-dual problem class (e.g. linear, quadratic, etc.) and to the application domain. More formally, we address the following matrix problem: $P = A \times H \times A^T$ where A is a sparse matrix and H is a block diagonal matrix with each diagonal block of H, H_i, having a rank-1 structure, i.e. $H_i = D_i + r_i * r_i^t$ for a diagonal matrix D_i and a column vector r_i.

The matrix triple product can be treated as two generic sparse matrix products by storing both A and its transpose and performing a full sweep through both instances. However, this generic approach places unnecessary demand on the memory systems and ignores the specialized structure of H. Instead, we introduce a single pass algorithm that exploits the block diagonal structure of the matrix H. While the one-phase scheme has an advantage over the two-phase scheme through exploiting the structure of matrix, the summation of sparse matrices becomes a bottleneck. Therefore, we propose a row-based one-phase scheme, where the summation of sparse matrices is replaced by the summation of sparse vectors, which can be computed efficiently using a sparse accumulator. We further improve the performance of the row-based one-phase scheme through the use of additional helper data structures.

2 Two Implementation Schemes: One-Phase vs. Two-Phase

2.1 Two-Phase Scheme

In this generic implementation scheme, the matrix triple product is treated as two generic sparse matrix products. That is, $Q = A \times H$, followed by $P = Q \times A^T$. In computing the product of two sparse matrices, the non-zero elements are stored in a column-compressed format. Then, the multiplication is conducted in an outer and an inner loop. The outer loop marches through the columns of the right matrix. The inner loop sweeps the columns of the left matrix and accumulates the products of their non-zero elements with the non-zero elements in the corresponding column of the right matrix. The summation of sparse vectors that forms a column of the product matrix can be efficiently performed using a sparse accumulator [2]. This generic approach places unnecessary demand on the memory systems since the intermediate product matrix $H \times A^T$ is explicitly computed and stored. Furthermore, it ignores the specialized structure inherent in H.

2.2 One-Phase Scheme

In this scheme, the product matrix is computed in one pass. First, the triple-product is decomposed according to blocks of matrix H as follows :

$$P = \sum_{i}^{\#\ of\ blocks\ in\ H} (P_i = A_i H_i A_i^T) \tag{2.1}$$

where H_i is an i-th block of H and A_i is a corresponding column block of A.

In equation-2.1, the summation of the sparse matrices with different non-zero distribution is not trivial and can be time consuming. To enhance the efficiency, we propose a row-based one-phase approach where the summation of sparse matrices is replaced by a sparse vector accumulator that composes each row (or column) of the product matrix, P, in order. By exploiting the structure of H, equation-2.1 may be computed as follows:

$$P = \sum_{i}^{\#\ of\ blocks\ in\ H} (A_i D_i A_i^T + (A_i r_i)(A_i r_i)^T) \tag{2.2}$$

From equation-2.2, the k-th row (or equivalently, column) of P, P_{k*}, is computed by the following equation:

$$
\begin{aligned}
P_{k*} &= \sum_{j}^{\#\ of\ columns\ in\ A} (A_{*j} d_j A_{*j}^T)_{k*} + \sum_{i}^{\#\ of\ non-unit\ blocks\ in\ H} (B_{*i} B_{*i}^T)_{k*} \\
&= \sum_j a_{kj} d_j A_{*j}^T + \sum_i b_{ki} B_{*i}^T \\
&= \sum_{j:a_{kj} \neq 0} a_{kj} d_j A_{*j}^T + \sum_{i:b_{ki} \neq 0} b_{ki} B_{*i}^T
\end{aligned}
\tag{2.3}
$$

where X_{i*} is i-th row of matrix X, X_{*i} is i-th column of matrix X, and $B_{*i} = A_i r_i$.

Furthermore, the computation of equation-2.3 is accelerated using the additional compressed sparse row index structures of matrices A and B which are stored in a compressed sparse column format. It is sufficient to construct a transpose of matrix A without its values since only the index structure is needed for this computation.

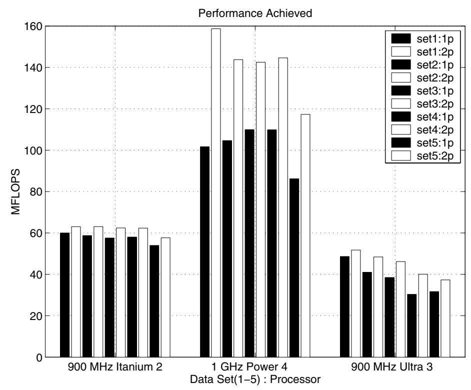
Finally, since the product matrix P is symmetric, we only compute the upper triangular part of the matrix by computing $a_{kj} d_j A_{k:m,j}^T$ and $b_{ki} B_{k:m,j}^T$, instead of $a_{kj} d_j A_{*j}^T$ and $b_{ki} B_{*j}^T$, where $X_{k:m,j}$ denotes k-th through m-th elements of the j-th column of matrix X. Accesses to elements A_{kj} are expedited by keeping an array of indices pointing to the next non-zero element in each column of A.

3 Performance and Modeling

We measure and compare performances of these two implementations using the five matrix sets summarized in the figure-1. This data set is obtained from real circuit design problems with progressively increasing sizes. The matrices grow sparser as their size increases. In the two rightmost columns, the table compares the number of floating-point operations and the memory requirements (dynamically allocated data size) of both schemes. These results show that both measures are almost halved in the one-phase scheme.

Set	# of rows in A	# of columns in A	# of NZs in A	# of NZs in H		# of fop.s	memory requirement
1	8648	42750	361K	195K	1-phase	11M	11M
					2-phase	24M	24M
2	14872	77406	667K	361K	1-phase	21M	20M
					2-phase	45M	41M
3	21096	112150	977K	528K	1-phase	31M	29M
					2-phase	66M	60M
4	39768	217030	1913K	1028K	1-phase	60M	57M
					2-phase	129M	118M
5	41392	244501	1633K	963K	1-phase	31M	50M
					2-phase	66M	113M

Fig. 1. Matrix sets used in the performance measurement



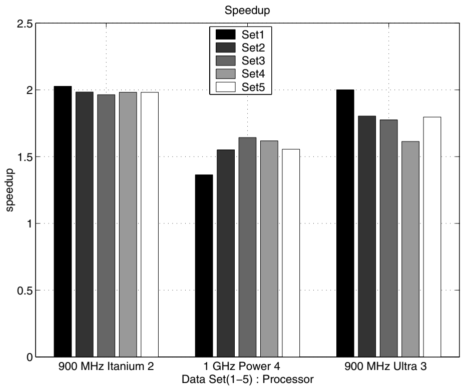


Fig. 2. Achieved Mflop rate and speedup The left graph shows the Mflop rate of the one-phase (1p) and two-phase (2p) schemes for the five matrix sets on an Itanium-2, a Power-4 and an Ultrasparc-3. The right graph shows the speedup of the one-phase scheme over the two-phase scheme for the same matrix sets on the 3 different platforms

The graphs in figure-2 show the Mflop rates (left) and speedups (right) of the one-phase scheme over the two-phase scheme for this data set on a 900 MHz Itanium-2, an 1 GHz Power-4, and a 900 MHz Ultrasparc-3. The speedup on the Itanium-2 is almost 2, and overall the speedup is over 1.5. The Mflop rate is lower in the one-phase scheme in all cases. The reduction in the execution time for the one-phase scheme is attributed to the reduced number of floating-point operations and memory accesses.

Figure-3 shows the pre-processing time on these three platforms. The pre-processing time is shown relative to the multiplication time of the two-phase scheme for each matrix. In the one-phase scheme, the pre-processing time is spent on

– counting the number of non-zeros in B and P to determine the amount of memory allocation,
– computing the structure of matrix B, and
– constructing the row-major structure of A and B.

In the two-phase scheme, it is spent on

– generating A^t, and
– counting the number of non-zeros in Q and P.

Figure-3 shows there is an added overhead of 1.25-1.75 in the one-phase scheme over relative to the two-phase scheme. Typically the triple matrix product is repeatedly computed 100-120 times, while the pre-processing time is spent only once. Hence, this added overhead is considered to be negligible.

We also modeled the execution of the one-phase and two-phase schemes. Figure-4 shows the measured and modeled execution time on the Itanium-2. The dominant factor in the one-phase scheme is due to accessing A's elements. The number of accesses is computed as follows:

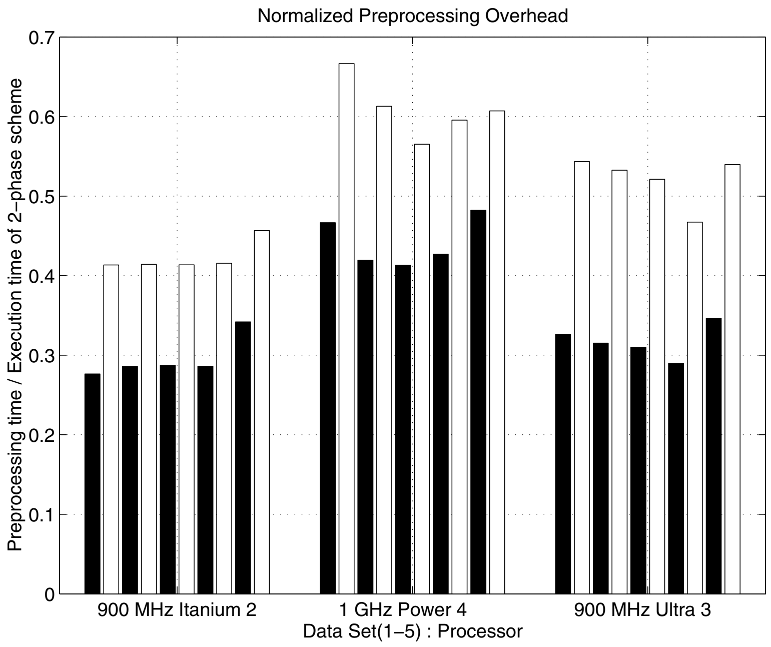


Fig. 3. Overhead of pre-processing The overheads of the one-time pre-processing in the one-phase and two-phase schemes are shown relative to the multiplication time of the two-phase scheme for each matrix on three platforms

$$\sum_{i}^{\#\ of\ columns\ in\ A}\ \sum_{k=1}^{nnz(A_{*i})} k + \sum_{i}^{\#\ of\ columns\ in\ B}\ \sum_{k=1}^{nnz(B_{*i})} k \tag{3.4}$$

The dominant factor in the two-phase scheme results from the access of elements of X in computing $X \times Y$:

$$\sum_{i}^{\#\ of\ columns\ in\ X} nnz(X_{*i})nnz(Y_{i*}) \tag{3.5}$$

This is computed for both products: $Q = H \times A^T$ and $P = A \times Q$. The measured execution times are between the lower and upper bounds of computed execution times with the modeled lower and upper bounds of the memory latency.

4 Conclusion

In our prior work [3,4,5] we have demonstrated the effectiveness of using data structure and algorithm transformations to significantly improve the performance of sparse matrix-vector multiplication.

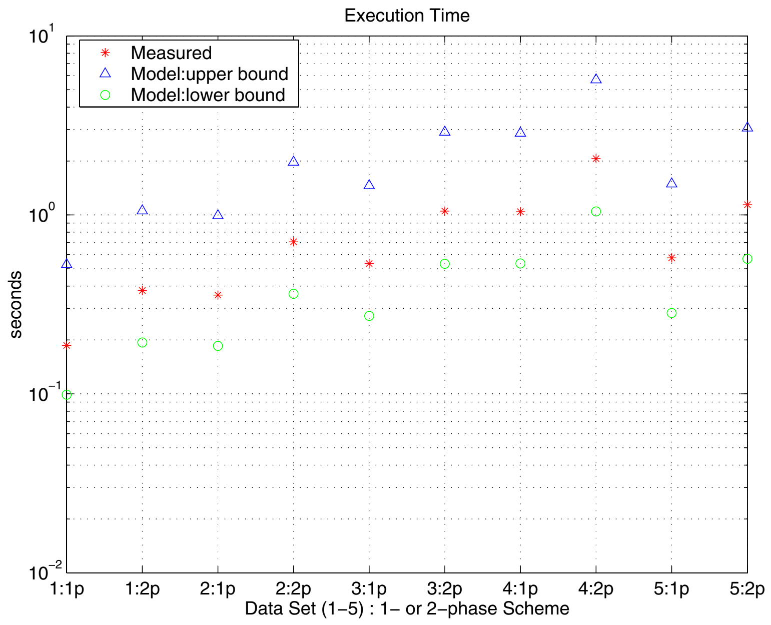


Fig. 4. Modeled and Measured Execution Time The execution time of the triple-product in the one-phase scheme is measured on a 900 MHz Itanium, and is shown with upper and lower bounds of computed execution time based on a memory latency model. The time is measured and modeled for the one-phase (1p) and two-phase (2p) schemes for the matrix data set in Figure-1

In this work we have examined an important sparse matrix kernel that commonly arises in solving primal-dual optimization problems. For the given data set, the speed-up of the one-phase scheme over the two-phase scheme is 2.04 on a 900 MHz Intel Itanium-2, 1.63 on a 1 GHz Power-4, and 1.99 on a 900 MHz Sun Ultra-3 platform. The sparse matrix triple product is a higher level kernel than those hitherto considered for automatic tuning in sparse matrix operations. The use of algebraic transformations to exploit the problem structure represents a higher level of tuning for domain-specific optimizations that can significantly improve performance for a wide class of problems.

Acknowledgment

We thank Richard Vuduc for his insightful discussion and for collecting measurement data on Power 4 and Ultra 3.

References

1. M.X. Goemans and D.P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. *Approximation Algorithms for NP-hard Problems*, pages 144–191, PWS Publishing Co., Boston, MA, 1996.

2. J.R. Gilbert, C. Moler and R. Schreiber. Sparse matrices in Matlab: Design and implementation. *SIAM J. Matrix Analysis and Applications*,13:333–356, 1992.
3. E. Im and K.A. Yelick. Optimizing Sparse Matrix Computations for Register Reuse in SPARSITY. In *Proceedings of the International Conference on Computational Science*, volume 2073 of *LNCS*, pages 127–136, San Francisco, CA, May 2001, Springer.
4. E. Im, K.A. Yelick and R Vuduc. SPARSITY: Framework for Optimizing Sparse Matrix-Vector Multiply. *International Journal of High Performance Computing Applications*, 18(1):135–158, February, 2004.
5. R. Vuduc, A. Gyulassy, J.W. Demmel and K.A. Yelick. Memory Hierarchy Optimizations and Bounds for Sparse $A^T Ax$. In *Proceedings of the ICCS Workshop on Parallel Linear Algebra*, volume 2660 of *LNCS*, pages 705–714, Melbourne, Australia, June 2003, Springer.

Adapting Distributed Scientific Applications to Run-Time Network Conditions*

Masha Sosonkina

Ames Laboratory and Iowa State University, Ames IA 50010
masha@scl.ameslab.gov

Abstract. High-performance applications place great demands on computation and communication resources of distributed computing platforms. If the availability of resources changes dynamically, the application performance may suffer, which is especially true for clusters. Thus, it is desirable to make an application aware of system run-time changes and to adapt it dynamically to the new conditions. We show how this may be done using a helper tool (middleware NICAN). In our experiments, NICAN implements a packet probing technique to detect contention on cluster nodes while a distributed iterative linear system solver from the pARMS package is executing. Adapting the solver to the discovered network conditions may result in faster iterative convergence.

1 Introduction

A typical high-performance scientific application places high demands on the computational power and communication subsystem of a distributed computing platform. It has been recorded [12] that cluster computing environments can successfully meet these demands and attain the performance comparable to supercomputers. However, because of the possibly heterogeneous nature of clusters, such performance enhancing techniques as load balancing or non-trivial processor mapping become of vital importance. In addition, the resources available to the application may vary dynamically at a run-time, creating imbalanced computation and communication. This imbalance introduces idle time on the "fastest" processors at the communication points after each computational phase. Taking into consideration an iterative pattern of computation/communication interchange, it could be beneficial for a distributed application to be aware of the dynamic system conditions present on the processors it is mapped too. Dynamic system conditions could include the load on the CPU, the amount of memory available, or the overhead associated with network communication. There have been many tools (e.g., [5,1]) created to help learn about the conditions present. One of the requirements for such tools is to provide an easy-to-use interface to scientific applications, which also *translates* and *filters* the low-level detailed system information into categories meaningful to applications. For a scientific application, in which the goal is to model and solve a physical problem rather

* This work was supported in part by NSF under grants NSF/ACI-0000443, NSF/ACI-0305120, and NSF/INT-0003274, in part by the Minnesota Supercomputing Institute, and in part by the U.S. Department of Energy under Contract W-7405-ENG-82.

than to investigate the computer system performance, such an interface is very important. It also constitutes a major reason why inserting "performance recording hooks" directly into application's code may not be viable for a wide range of applications and for a multitude of computing system parameters. Thus the usage of a helper middleware is justified. When used at application's run-time, the middleware must be light-weight contrary to typical throughput benchmarks or operating system calls, which may heavily compete with the application for network or system resources.

In this paper (Section 3) we outline a network probing technique that, by means of sending carefully chosen trains of small packets, attempts to discover network contention without exhibiting the overhead inherent to network throughput benchmarks [9]. A light-weight technique may be used *simultaneously* with the computational stage of a scientific application to examine the communication subsystem without hindering the application performance. Section 2 presents a brief description of a proposed earlier framework that enables adaptive capabilities of scientific applications during the run-time. In Section 4, we consider a case study of determining dynamic network conditions while a parallel linear system solution solver pARMS is executing. A short summary is provided in Section 5.

2 Enabling Runtime Adaptivity of Applications

Network Information Conveyer and Application Notification (NICAN) is a framework which enables adaptation functionality of distributed applications [10]. The main idea is to decouple the process of analyzing network information from the execution of the parallel application, while providing the application with critical network knowledge in a timely manner. This enables non-intrusive interaction with the application and low overhead of the communication middleware. NICAN and the application interact according to a register and notify paradigm: the application issues a request to NICAN specifying the parameters it is interested in, and NICAN informs the application of the critical changes in these parameters. The application adaptation may be triggered by NICAN when certain resource conditions are present in the system. When the distributed application starts executing, each process starts a unique copy of NICAN as a child thread. This implementation is particularly useful in a heterogeneous environment because there is no requirement that the NICAN processes be homogeneous or even be running on the same type of machine. The process of monitoring a requested network parameter is separated from the other functions, such as notification, and is encapsulated into a module that can be chosen depending on the network type, network software configuration, and the type of network information requested by the application. Figure 1 depicts NICAN's functionality as the interaction of four major entities.

NICAN reads the resource monitoring requests using an XML interface, which enables diverse specifications and types of the application requests. Another functionality of NICAN is to call the adaptation functions provided when appropriate. If a particular resource condition is never met, then the adaptation is never triggered and the application will proceed as if NICAN had never been started. To make NICAN versatile and to provide a wide variety of monitoring capabilities, it is driven by dynamically loaded modules.

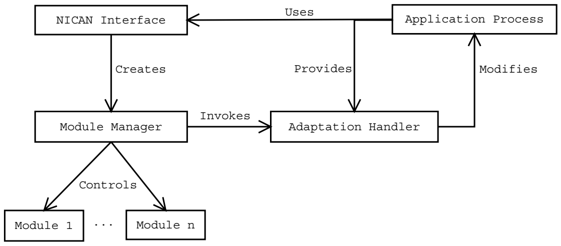


Fig. 1. Interaction of NICAN's components

```
/* Include the NICAN header file */
#include <nican.h>
/* The handlers are declared in the global scope */
void HandlerOne(const char* data) {/* empty */};
void HandlerTwo(const char* data) {/* empty */};
/* The application's main function */
void main() {
    const char xmlFile[] = "/path/to/xmlFile";
    /* Start NICAN's monitoring */
    Nican_Initialize(xmlFile,
                     2,
                     "HandlerOne", &HandlerOne,
                     "HandlerTwo", &HandlerTwo );
    /* Application code runs while NICAN operates */
    /* ... */
    /* Terminate NICAN's monitoring */
    Nican_Finalize(); }
```

Fig. 2. A trivial example of how to use NICAN

Figure 2 demonstrates how easy it is for the application to use NICAN. First the application must include a header file with the declarations for the NICAN interface functions. There are two adaptation handlers specified, *HandlerOne* and *HandlerTwo*, which for this trivial example are empty functions. The path to the XML file is specified and passed as the first parameter to *Nican_Initialize*. The application is informing NICAN of two adaptation handlers. *Nican_Initialize* will start the threads required for NICAN to operate and return immediately, allowing it to run simultaneously with the application. When the application is finished, or does not require the use of NICAN any longer, it calls the *Nican_Finalize* function, which returns after all the threads related to NICAN have been safely terminated.

3 Packet Probing to Examine Communication Overhead

In a cluster, we can consider two types of network transmissions following the terminology given in [2]. One type is *latency bound* transmission, and the other is a *bandwidth bound* transmission. A latency bound transmission is one where the transmission time required is dependent only on a one-time cost for processing any message. By using very small messages, on the order of a few hundred bytes, the cost of processing the message is reduced to a minimum. A bandwidth bound transmission is one where the time required is dependent on not just a one-time cost, but also the bandwidth available (see e.g., [9]). Typically bandwidth bound transmissions are comprised of large messages, which cause additional overhead while those messages are queued on network interfaces and processed. Latency bound transmissions have the attractive property that they do not cause a significant software overhead on the protocol stack of the machine processing the message. Thus, latency bound transmissions may be used as a means of communication subsystem performance analysis at application's runtime. This can be done while a distributed application performs its computational phase so as to not interfere with the application's communications. To better capture what conditions are present on a neighboring node, we use a train of packets rather than only two. This allows more time for the conditions to influence our train and is a trade-off between only two packets and flooding the network. We can use the notion of initial gap similar to the LogP model [3] to describe what happens to the probing packets. By introducing different types of load on the source and sink nodes we can affect the gaps recorded at the sink in a way that can be used to determine what type of load is present. The two metrics we will be using are the average gaps recorded at the sink and the standard deviation of those gaps.

3.1 Example of Probing: Fast Ethernet Networks

The cluster for our experiments is a collection of nine dual-processor nodes connected via 100Mbps Ethernet links by means of a shared hub. We send a train of packets directly from the source to the sink and perform timing on the sink. We vary the size of the packets across trials, but keep the number of packets constant. For each packet size we sent 100 trains of 64 packets and computed the average arrival time and the standard deviation for the distribution of arrival times. Note that 100 trains would be too many to be used in actual probing, as it is too close to "flooding" the network, but was used in our derivation of the technique described in this section. The actual amount of probing will depend on the time warranted by the computational phase of the application. We have conducted experiments that indicate using a single train of 32 packets may be sufficient. To add an additional load on the source and sink, we generated 30Mbps UDP network transmissions and external 98% CPU and memory loads on a node. Although we have performed a series of experiments to detect possible `(external load, competing flow)` combinations, we present here only a few illustrative cases. For more details see [11]. In particular, Figure 3 shows the case when no adverse conditions are present on the nodes. As packets grow larger they introduce a larger load on the network stacks. This is clearly the case with Figure 3 depicting a distinct *bend* formed at the point where the probes are 400B in size. We can approximate this point using the rate of increase for the arrival gaps as the probes get larger. A different situation is shown in Figure 4 where there is no pronounced bend. The "strength" of this bend can be used to help classify what conditions may be present. The location of the bend can be determined during the first set of probes, or by executing the proposed algorithm on a given computing system *a priori* before the actual application is run. Once this point is known we can narrow the scope of the probe sizes to reduce the impact and time required for the dynamic analysis.

In each experiment the gap introduced on the source between the packets, at the user level, was constant with very little deviation. Therefore any deviation measured at the

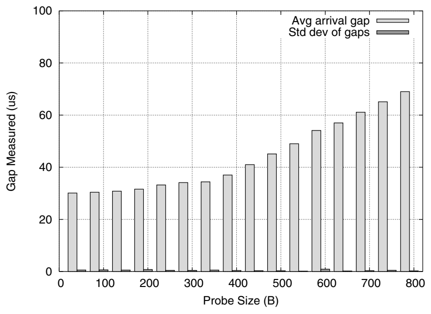


Fig. 3. No competing CPU load or network flows on either node

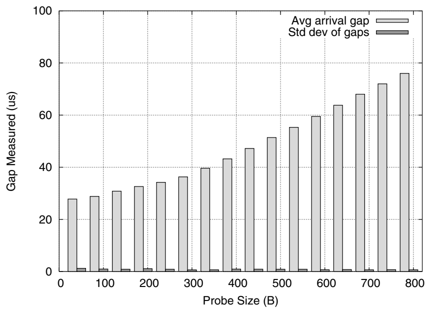


Fig. 4. No CPU load and 30 Mbps flow leaving the source node

sink was due to an external influence on our probes. Depending on what type of influence is placed on the source and sink we observe different amounts of deviation, which we use as the second factor to classify the network conditions present. In particular, for a given network type, we have constructed a decision tree (see [11]), tracing which one may detect seven possible cases of network or CPU related load on the source and sink nodes based on recorded deviations and average gap measurements.

4 Case Study: Runtime Changes in Communication Overhead of pARMS

pARMS is a parallel version of the Algebraic Recursive Multilevel Solver (ARMS) [8] to solve a general large-scale sparse linear system $Ax = b$, where A is a constant coefficient matrix, x is a vector of unknowns, and b is the solution vector. To solve such a system iteratively, one *preconditions* the system of equations into a form that is easier to solve. A commonly used (parallel) preconditioning technique, due to its simplicity, is Additive Schwarz procedure (see, e.g, [7]). In the iteration i, $i = 1, \ldots, m$, given the current solution x_i, Additive Schwarz computes the residual error $r_i = b - Ax_i$. Once r_i is known, δ_i is found by solving $A\delta_i = r_i$. To obtain the next iterate x_{i+1}, we simply compute $x_{i+1} = x_i + \delta_i$ and repeat the process until $|x_{i+1} - x_i| < \epsilon$, where ϵ is a user defined quantity. The Additive Schwarz procedure was used for all the experiments discussed here. To solve linear systems on a cluster of computers it is common to partition the problem using a graph partitioner and assign a subdomain to each processor. Each processor then assembles only the local equations associated with the elements assigned to it.

Our motivation for using packet probing is to find congested links in the underlying network of a cluster and to alert pARMS whenever its own adaptive mechanisms need to be invoked. To achieve this goal we have developed a module for NICAN that performs packet probing using the techniques described in Section 3. The design of MPI [4] allows pARMS to start a unique instance of NICAN in each task, each of which sends probes independently. Discovering network overhead on neighboring nodes has proven useful [6] for the overall performance of pARMS. Thus, ideally, we wish to have each node learn the conditions of the entire system.

We will demonstrate this NICAN module using a four-node pARMS computation, with each node probing a fifth node free of any network overhead. By using the various options provided by the module this situation is easily created using an XML file. The 4pack cluster with 32 dual Macintosh G4 nodes, located at the Scalable Computing Laboratory in Iowa State University, has been used for the experiments. Sixteen 4pack nodes have a single 400MHz CPU and the remaining have dual 700MHz CPUs. We used the faster nodes with Fast Ethernet interconnection for both our probes and MPI traffic because an implementation of NICAN packet probing module on Ethernet networks is already available. Figure 5 illustrates how the experiment is laid out. The squares represent the nodes used, with the label indicating the hostname of the machine. pARMS is mapped to `node0`,. . .,`node3`; the competing flows (called `Iperf traffic`) are entering node `iperf_dest`; and the probes sent from the pARMS nodes, are sinking into `probe_sink`.

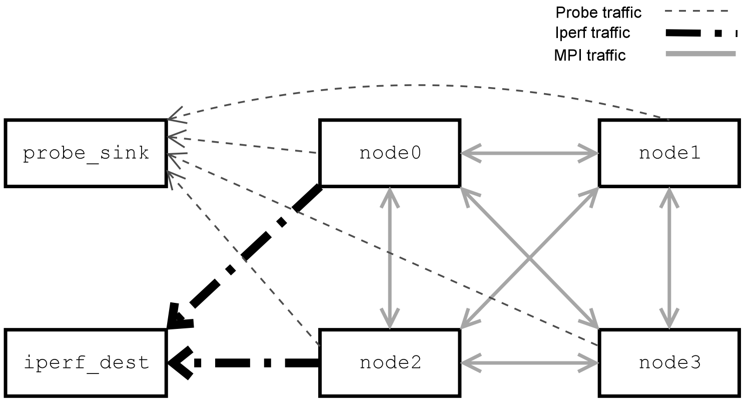


Fig. 5. Cluster node interaction used for the experiments

Consider the elliptic partial differential equation (PDE)

$$-\Delta u + 100\frac{\partial}{\partial x}\left(e^{xy}u\right) + 100\frac{\partial}{\partial y}\left(e^{-xy}u\right) - 1,000u = f \qquad (4.1)$$

solved on a two-dimensional rectangular (regular) grid with Dirichlet boundary conditions. It is discretized with a five-point centered finite-difference scheme on a $n_x \times n_y$ grid, excluding boundary points. The mesh is mapped to a virtual $p_x \times p_y$ grid of processors, such that a subrectangle of $r_x = n_x/p_x$ points in the x direction and $r_y = n_y/p_y$ points in the y direction are mapped to a processor. In the following experiments, the mesh size in each processor is kept constant at $r_x = r_y = 40$. In our experiments, four processors $(p_x = p_y = 2)$ have been used, thus resulting in a problem of the total size 6,400. This problem is solved by FGMRES(20) using Additive Schwarz pARMS preconditioning with one level of overlap and four inner iterations needed to solve the local subproblem with GMRES preconditioned with ILUT (see [7]).

Each MPI node will use NICAN's packet probing module to send probing packets to a fifth node known to be free of any network load. Then, at pARMS runtime, any network flow detected is indicative of a load on a node involved in the computation. The experimental procedure is outlined in Table 1. The "Phase" column represents the different phases of the experiment an the "Conditions present" column details what conditions were present in addition to the pARMS-related traffic. Table 1 also lists the average times required to complete the distributed sparse matrix-vector multiplication (called SpMxV) during each phase of this experiment (see columns `node0`,. . .,`node3`). The impact of competing network traffic is evident in how the unaffected nodes spend more time completing SpMxV. The extra time is actually accrued while they wait for the node affected by the network flow to transmit the required interface variables. The affected node does not perceive as long a waiting time because when it finally requests the interface unknowns from a neighbor, that neighbor can immediately send them. When we introduce the two network flows at the phase p_2, both `node0` and `node2` experience less waiting time, but we can also see how much impact competing network traffic can exert on pARMS. The average waiting time for unaffected nodes in this case has nearly tripled compared with p_0, increasing the overall solution time as a result. We only show the times for SpMxV because that is the specific source of increased execution time when additional network traffic is exerted upon a node in the computation. Once we are able to detect and monitor how much waiting time is incurred by each node a suitable adaptation can be developed to help balance the computation, and improve the performance in spite of competing network traffic [6]. The gist of the pARMS adaptation procedure is to *increase* the number of inner Additive Schwarz iterations performed locally in the *fast* processors, i.e., in those processors that incur idle time the most. For example, `node1` and `node3` are such processors in phase p_2. The amount of increase is determined experimentally and may be adjusted on subsequent outer iterations, such that the pARMS execution is kept balanced. With more inner iterations, the accuracy of the local solution becomes higher and will eventually propagate to the computation of the overall solution in an outer (global) iteration, thus reducing the *total number* of outer iterations. Figure 6, taken from [6] for the measurements on an IBM SP, shows the validity of suggested pARMS adaptation. In particular, with the increase of the number of inner iterations, the waiting time on all the processors becomes more balanced, while the total solution time and the number of outer iterations decrease. Figures 7 and 8 demonstrate how the gaps recorded on the nodes are used to determine the conditions present. In Figure 7 the bend discussed in Section 3 is clearly visible when the probes transition from 350B to 400B. Also, because we are using only 32 probing packets the deviation is larger than that observed in Figure 3. In Figure 8 we see the effect that the competing flow has on the probing packets. The distinct bend is no longer present, and by tracing through the decision tree corresponding to the given network type, we can determine that there is a competing flow leaving the source. We only illustrate the results for these two cases but similar plots can be constructed to show how the other conditions are detected.

Table 1. Phases of network conditions and average times (s) for SpMxV at each phase during pARMS execution

Phase	Conditions present	node0	node1	node2	node3
p_0	No adverse conditions present	.0313	.0330	.0331	.0398
p_1	40 Mbps flow from node0	.0293	.0698	.0899	.0778
p_2	40 Mbps flows from node0 and node2	.0823	.1089	.0780	.1157
p_3	40 Mbps flow from node2	.0668	.0847	.0291	.0794
p_4	No adverse conditions present	.0293	.0319	.0474	.0379

Adapt. yes/no	Outer Iter.	Solution, s
no	5,000	4,887.78
yes	432	398.67

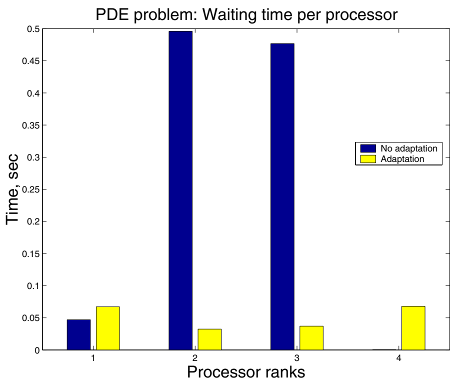


Fig. 6. Adaptation of pARMS on a regular-grid problem

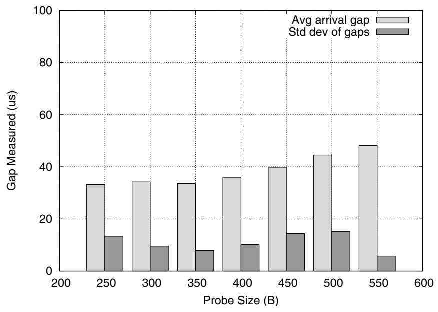


Fig. 7. The probe gaps observed by node0 at phase p_0 in Table 1

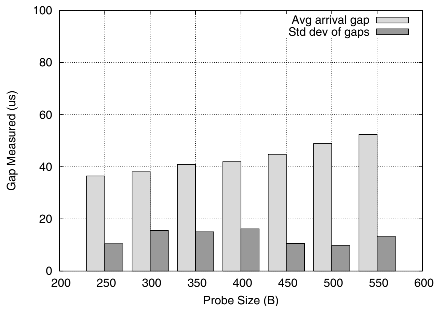


Fig. 8. The probe gaps observed by node0 at phase p_1 in Table 1

5 Conclusions

We have described a way to make distributed scientific applications network- and system-aware by interacting with an easy-to-use external tool rather than by obtaining and processing the low-level system information directly in the scientific code. This approach is rather general, suiting a variety of applications and computing platforms, and causes no excessive overhead. The case study has been presented in which the NICAN middleware serves as an interface between parallel Algebraic Recursive Multilevel Solver (pARMS) and the underlying network. We show how a light-weight packet probing technique is used by NICAN to detect dynamically network contention. In particular, NICAN is able to detect and classify the presence of competing flows in the nodes to which pARMS is mapped. Upon this discovery, pARMS is prompted to engage its own adaptive mechanisms leading to a better parallel performance.

References

1. D. Andersen, D. Bansal, D. Curtis, S. Seshan, and H. Balakrishnan. System support for bandwidth management and content adaptation in Internet applications. In *Proceedings of 4th Symposium on Operating Systems Design and Implementation*, pages 213–226, 2000.
2. C. Bell, D. Bonachea, Y. Cote, J. Duell, P. Hargrove, P. Husbands, C. Iancu, M. Welcome, and K. Yelick. An evaluation of current high-performance networks. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003.
3. D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a realistic model of parallel computation. In *Principles Practice of Parallel Programming*, pages 1–12, 1993.
4. Message Passing Interface Forum. MPI: A message-passing interface standard. Technical Report Computer Science Department Technical Report CS-94-230, University of Tennessee, Knoxville, TN, May 5 1994.
5. J. Hollingsworth and P. Keleher. Prediction and adaptation in Active Harmony. *Cluster Computing*, 2(3):195–205, 1999.
6. D. Kulkarni and M. Sosonkina. A framework for integrating network information into distributed iterative solution of sparse linear systems. In José M. L. M. Palma, et al. editors, *High Performance Computing for Computational Science - VECPAR 2002, 5th International Conference, Porto, Portugal, June 26-28, 2002, Selected Papers and Invited Talks*, volume 2565 of *Lecture Notes in Computer Science*, pages 436–450. Springer, 2003.
7. Y. Saad. *Iterative Methods for Sparse Linear Systems, 2nd edition*. SIAM, Philadelpha, PA, 2003.
8. Y. Saad and B. Suchomel. ARMS: An algebraic recursive multilevel solver for general sparse linear systems. Technical Report Minnesota Supercomputing Institute Technical Report umsi-99-107, University of Minnesota, 1999.
9. Q. Snell, A. Mikler, and J. Gustafson. NetPIPE: A network protocol independent performance evaluator. In *IASTED International Conference on Intelligent Information Management and Systems*, June 1996.
10. M. Sosonkina and G. Chen. Design of a tool for providing network information to distributed applications. In *Parallel Computing Technologies PACT2001*, volume 2127 of *Lecture Notes in Computer Science*, pages 350–358. Springer-Verlag, 2001.
11. S. Storie and M. Sosonkina. Packet probing as network load detection for scientific applications at run-time. In *IPDPS 2004 proceedings*, 2004. 10 pages.
12. Top 500 supercomputer sites. http://www.top500.org/.

Sparse Direct Linear Solvers: An Introduction

Organizer: Sivan Toledo

Tel-Aviv University, Israel
stoledo@tau.ac.il

Introduction

The minisymposium on sparse direct solvers included 11 talks on the state of the art in this area. The talks covered a wide spectrum of research activities in this area. The papers in this part of the proceedings are expanded, revised, and corrected versions of some the papers that appeared in the CD-ROM proceedings that were distributed at the conference. Not all the talks in the minisymposium have corresponding papers in these proceedings.

This introduction explains the significance of the area itself. The introduction also briefly presents, from the personal and subjective viewpoint of the organizer, the significance and contribution of the talks.

Sparse direct linear solvers solve linear systems of equations by factoring the sparse coefficient matrix into a product of permutation, triangular, and diagonal (or block diagonal) matrices. It is also possible to factor sparse matrices into products that include orthogonal matrices, such as sparse QR factorizations, but such factorizations were not discussed in the minisymposium. Sparse direct solvers lie at the heart of many software applications, such as finite-elements analysis software, optimization software, and interactive computational engines line Matlab and Mathematica.

For most classes of matrices, sparse direct linear solvers scale super-linearly. That is, the cost of solving a linear system with n unknowns grows faster than n. This has led many to search for alternatives solvers with better scaling, mostly in the form of iterative solvers. For many classes of problems, there are now iterative solvers that scale better than direct solvers, and iterative solvers are now also widely deployed in software applications. But iterative solvers have not completely displaced iterative solvers, at least not yet. For some classes of problems, fast and reliable iterative solvers are difficult to construct. In other cases, the size and structure of linear system that application currently solve are such that direct solvers are simply faster. When applications must solve many linear systems with the same coefficient matrix, the amortized cost of the factorization is low. As a result of these factors, sparse direct solvers remain widely used, and research on them remains active.

The talk that I think was the most important in the minisymposium was delivered by Jennifer Scott, and was based on joint work with Nick Gould and Yifan Hu. Jennifer's talk, on the evaluation of sparse direct solvers for symmetric linear systems, described a large-scale study in which she and her colleagues carefully compared several solvers. A comprehensive, objective, and detailed comparison of existing techniques (concrete software packages in this case) is an essential tool for both researchers working within the field and users of the technology. But such studies are difficult to carry out, and are not as exciting to conduct as research on new techniques. I am, therefore, particularly grateful to Scott, Gould, and Hu for carrying out and publishing this important research. The significance of this research reaches beyond direct solvers, because iterative solvers can be included in similar studies in the future.

Two talks discussed pre-orderings for sparse direct solvers. Pre-ordering the rows and/or columns of a sparse matrix prior to its factorization can reduce the fill in the factors and the work required to factor them. Pre-ordering also affect the parallelism and locality of reference in the factorization. Yngve Villanger presented algorithms that can improve a given symmetric ordering, developed jointly with Pinar Heggernes. Yifan Hu presented work that he has done with Jennifer Scott on column pre-ordering for partial-pivoting unsymmetric solvers, with emphasis on generating orderings that lead to effective parallel factorizations.

Xiaoye Li talked about modeling the performance of the parallel symbolic factorization phase of SuperLU_DIST, a distributed-memory partial-pivoting unsymmetric solver. This work was conducted together with Laura Grigori.

Two talks focused on solvers for symmetric but indefinite linear systems. Stefan Roöllin described his work with Olaf Schenk on using maximum-weight matching to improve the performance of such solvers. Sivan Toledo gave a talk on an out-of-core implementation of a more conventional symmetric indefinite solver; this work was done with Omer Meshar.

Jose Herrero presented surprising results on a sparse hypermatrix Cholesky factorization, work that he has done together with Juan Navarro. The hypermatrix representation is a data structure for storing and operating on sparse matrices. Herrero and Navarro showed that at least on some classes of matrices, a solver that uses a hypermatrix representation outperforms a more conventional sparse direct solver that is based on a supernodal partitioning of the matrix.

The last three talks in the minisymposium form some sort of a group, in that all three gave a fairly high-level overview of one specific sparse direct solver. Anshul Gupta talked about WSMP, a solver that can now solve both symmetric and unsymmetric linear systems and that can exploit both shared-memory and distributed-memory parallelism (in the same run). Klaus Gaertner talked about PARDISO, which has been developed together with Olaf Schenk. In his talk, Klaus emphasized the need to exploit application-specific properties of matrices and gave examples from matrices arising in circuit simulations. Florin Dobrian talked about OBLIO, an object-oriented sparse direct solver that he developed with Alex Pothen. Florin's talk focused mainly on the software-engineering aspects of sparse direct solvers.

Oblio: Design and Performance

Florin Dobrian and Alex Pothen[⋆]

Department of Computer Science and Center for Computational Sciences
Old Dominion University
Norfolk, VA 23529, USA
{dobrian,pothen}@cs.odu.edu

Abstract. We discuss Oblio, our library for solving sparse symmetric linear systems of equations by direct methods. The code was implemented with two goals in mind: efficiency and flexibility. These were achieved through careful design, combining good algorithmic techniques with modern software engineering. Here we describe the major design issues and we illustrate the performance of the library.

1 Introduction

We describe the design of Oblio, a library for solving sparse symmetric linear systems of equations by direct methods, and we illustrate its performance through results obtained on a set of test problems.

Oblio was designed as a framework for the quick prototyping and testing of algorithmic ideas [3] (see also [1,5,10,11] for recent developments in sparse direct solvers). We needed a code that is easy to understand, maintain and modify, features that are not traditionally characteristic of scientific computing software. We also wanted to achieve good performance. As a consequence, we set two goals in our design: efficiency and flexibility.

Oblio offers several options that allow the user to run various experiments. This way one can choose the combination of algorithms, data structures and strategies that yield the best performance for a given problem on a given computational platform. Such experimentation is facilitated by decoupled software components with clean interfaces. The flexibility determined by the software design is Oblio's major strength compared to other similar packages.

Out of the three major computational phases of a sparse direct solver, *preprocessing*, *factorization* and *triangular solve*, only the last two are currently implemented in Oblio. For preprocessing we rely on external packages.

2 Design

Factorization Types. The first major choice in Oblio is the factorization type. Generally, the direct solution of a linear system $Ax = b$ requires factoring a permutation of the

[⋆] This research was partially supported by U.S. NSF grant ACI-0203722, U.S. DOE grant DE-FC02-01ER25476 and LLNL subcontract B542604.

coefficient matrix A into a product of lower and upper triangular matrices (factors) L and U (L unit triangular). If A is symmetric then U can be further factored as DL^T, where D is generally block diagonal, and if A is positive definite as well then we can write $A = LU = LDL^T = \tilde{L}\tilde{L}^T$ (D is purely diagonal in this case), where $\tilde{L}$ is lower triangular. The latter decomposition of A is the Cholesky factorization and $\tilde{L}$ is the Cholesky factor.

In Oblio we offer three types of factorizations. The first one is the Cholesky factorization $\tilde{L}\tilde{L}^T$, which does not require pivoting and therefore has a static nature. The other two are more general LDL^T factorizations. One of them is static, the other one is dynamic.

The static LDL^T factorization does not usually allow row/column swaps (*dynamic pivoting*). Instead, it relies on small perturbations of the numerical entries (*static pivoting*) [7]. Although this changes the problem, it is possible to recover solution accuracy through *iterative refinement*. The static LDL^T factorization can be, however, enhanced, by allowing dynamic pivoting as long as this does not require modifications of the data structures. In Oblio we provide the framework for both approaches.

Numerical preprocessing is generally required before a static LDL^T factorization in order to reduce the number of perturbations as well as the number of steps of iterative refinement [4]. Since the numerical preprocessing of sparse symmetric problems is currently an area of active research, this type of preprocessing is not yet present in Oblio.

The dynamic LDL^T factorization performs dynamic pivoting using 1×1 and 2×2 pivots and can modify the data structures. Two pivot search strategies are currently available, one biased toward 1×1 pivots and one biased toward 2×2 pivots [2]. Note that the dynamic LDL^T factorization can also benefit from numerical preprocessing: a better pivot order at the beginning of the factorization can reduce the number of row/column swaps.

For positive definite matrices the best choice is the $\tilde{L}\tilde{L}^T$ factorization. For matrices that are numerically more difficult one must choose between the two LDL^T factorizations. The dynamic LDL^T factorization is more expensive but also more accurate. It should be used for those matrices that are numerically the most difficult. For less difficult matrices the static LDL^T factorization might be a better choice.

Factorization Algorithms. The second major choice in Oblio is the factorization algorithm. We offer three major column based supernodal algorithms: *left-looking*, *right-looking* and *multifrontal*.

These three factorization algorithms perform the same basic operations. The differences come from the way these operations are scheduled. In addition to coefficient matrix and factor data these algorithms also manage temporary data. For left-looking and right-looking factorization this is required only for better data reuse, but for multifrontal factorization temporary data are also imposed by the way the operations are scheduled.

The multifrontal algorithm generally requires more storage than the other two algorithms and the difference comes from the amount of temporary data. If this is relatively small compared to the amount of factor data then the multifrontal algorithm is a good choice. The right-looking and multifrontal algorithms perform the basic operations at a coarser granularity than the left-looking algorithm, which increases the potential for data reuse. The right-looking algorithm is not expected to perform well in an out-of-core context [9].

Factor Data Structures. The third major choice in Oblio is the factor data structure (for $\tilde{L}\tilde{L}^T$ factorization this stores $\tilde{L}$; for LDL^T factorization this stores L and D together). A static factorization can be implemented with a static data structure while a dynamic factorization requires a dynamic data structure. We offer both alternatives, although a static factorization can be implemented with a dynamic data structure as well.

The difference between the two data structures comes from the way storage is allocated for supernodes. In a static context a supernode does not change its dimensions and therefore a large chunk of storage can be allocated for all supernodes. In a dynamic context a supernode can change its dimensions (due to delayed columns during pivoting) and in this case storage needs to be allocated separately for each supernode. In order to access the data we use the same pointer variables in both cases. This makes the data access uniform between implementations.

The two data structures manage in-core data and therefore we refer to them as *in-core static* and *in-core dynamic*. In addition, we offer a third, *out-of-core*, data structure that extends the in-core dynamic data structure. The out-of-core data structure can store the factor entries both internally and externally. Of course, most of the factor entries are stored externally. Data transfers between the two storage layers are performed through I/O operations at the granularity of a supernode. A supernode is stored in-core only when the factorization needs to access it.

Other Features. A few other features are of interest in Oblio. By using a dynamic data structure for the elimination forest [8] it is easy to reorder for storage optimization and to amalgamate supernodes for faster computations. Iterative refinement is available in order to recover solution accuracy after perturbing diagonal entries during a static LDL^T factorization or after a dynamic LDL^T factorization that uses a relaxed pivoting threshold. Oblio can also factor singular matrices and solve systems with multiple right hand sides.

Implementation. Oblio is written in C++ and uses techniques such as dynamic memory allocation, encapsulation, polymorphism, and templates. Dynamic memory allocation is especially useful for temporary data. In order to minimize the coding effort we combine it with encapsulation most of the time, performing memory management within constructors and destructors. We use polymorphism in order to simplify the interaction between different factorization algorithms and different factor data structures, and we rely on templates in order to instantiate real and complex valued versions of the code.

Oblio is organized as a collection of *data* and *algorithm* classes. Data classes describe passive objects such as coefficient matrices, vectors, permutations and factors. Algorithm classes describe active objects such as factorizations and triangular solves. The major classes and the interactions between them are depicted in Fig. 1 (real valued only).

We discuss now the interaction between the three factorization algorithms and the three factor data structures in more detail. Generally we would require three different implementations for each algorithm and thus a total of nine implementations. That would determine a significant programming effort, especially for maintaining the code. We rely

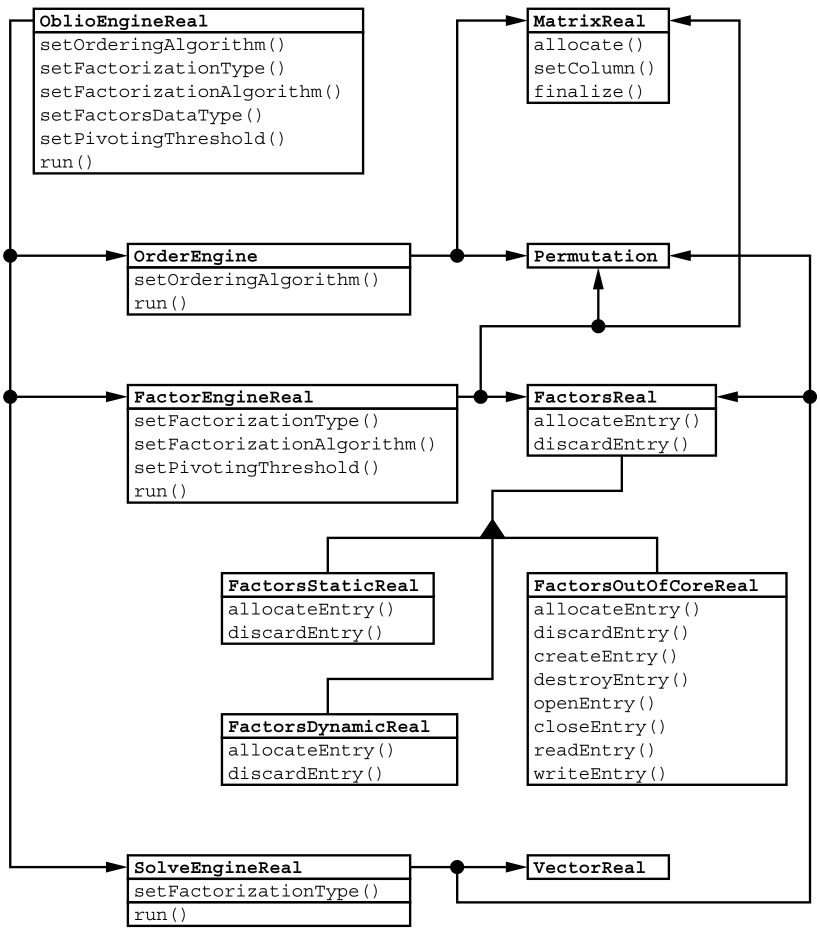


Fig. 1. The major classes from Oblio and the interactions between them (only the real valued numerical classes are shown but their complex valued counterparts are present in Oblio as well). The factorization algorithms interact with the abstract class `FactorsReal`. The actual class that describes the factors can be `FactorsStaticReal`, `FactorsDynamicReal` or `FactorsOutOfCoreReal` (all three derived from `FactorsReal`), the selection being made at run time

on polymorphism instead and use only three implementations, one for each algorithm. Remember that the difference between the two in-core data structures comes from the supernode storage allocation. But as long as we can use a common set of pointers in order to access data, a factorization algorithm does not need to be aware of the actual storage allocation. Therefore the factorization algorithms interact with an abstract factor class that allows them to access factor data without being aware of the particular factor implementation. The whole interaction takes place through abstract methods.

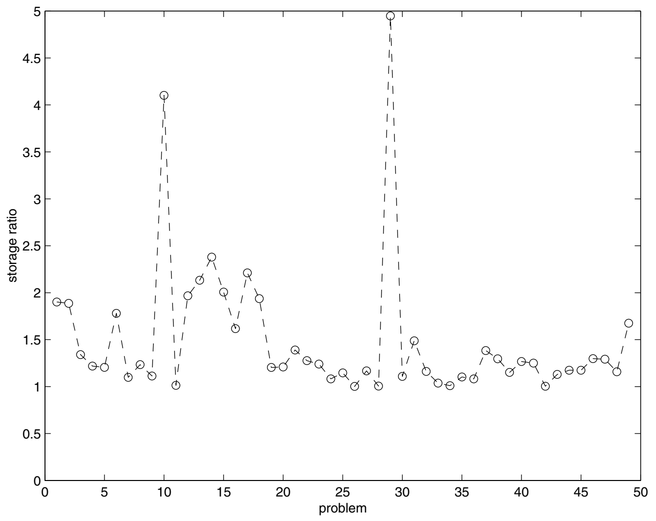


Fig. 2. The ratio between the storage required by the multifrontal algorithm and the storage required by the left-looking and right-looking algorithms, $\tilde{L}\tilde{L}^T$ factorization

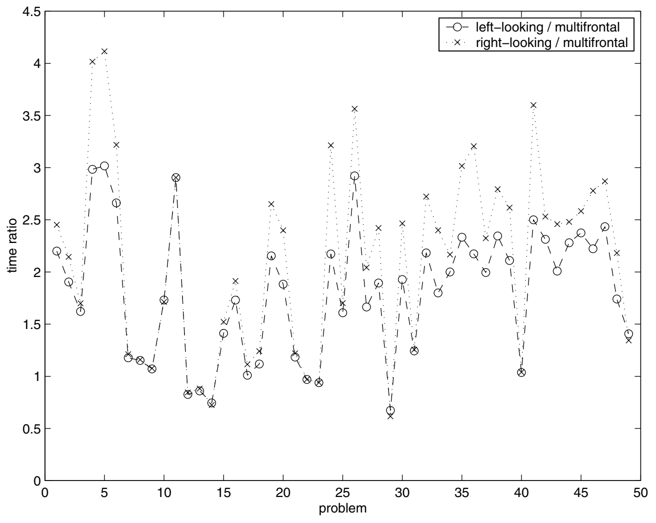


Fig. 3. The ratios between the execution times of the left-looking and multifrontal algorithms, and between the execution times of the right-looking and multifrontal algorithms, $\tilde{L}\tilde{L}^T$ factorization (IBM Power 3 platform: 375 MHz, 16 GB)

For out-of-core factorization we make a trade-off. In addition to the basic operations that are required by an in-core factorization, an out-of-core factorization requires I/O operations. These can be made abstract as well but that would not necessarily be an elegant software design solution. Another solution, requiring additional code but more

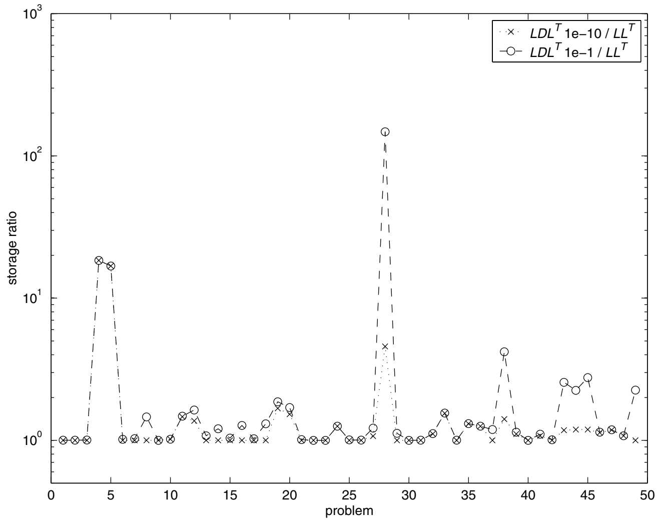


Fig. 4. The ratio between the storage required by the dynamic LDL^T factorization and the storage required by the $\tilde{L}\tilde{L}^T$ factorization, using the multifrontal algorithm, for pivoting thresholds of 1e-1 and 1e-10, respectively

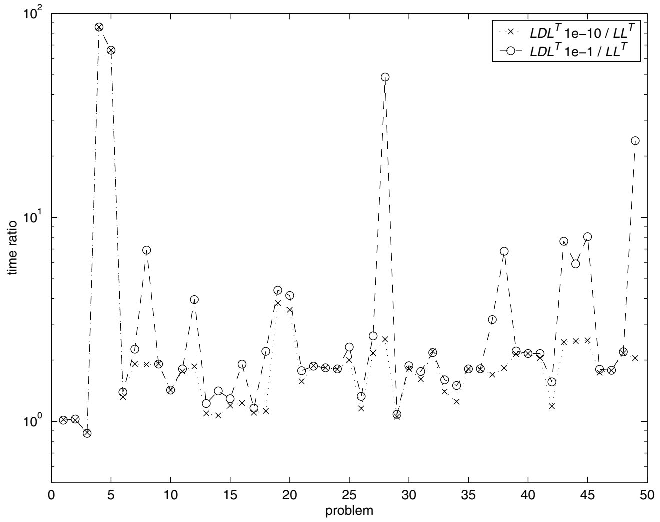


Fig. 5. The ratio between the execution time of the dynamic LDL^T factorization and the execution time of the $\tilde{L}\tilde{L}^T$ factorization, using the multifrontal algorithm, for pivoting thresholds of 1e-1 and 1e-10, respectively (IBM Power 3 platform: 375 MHz, 16 GB)

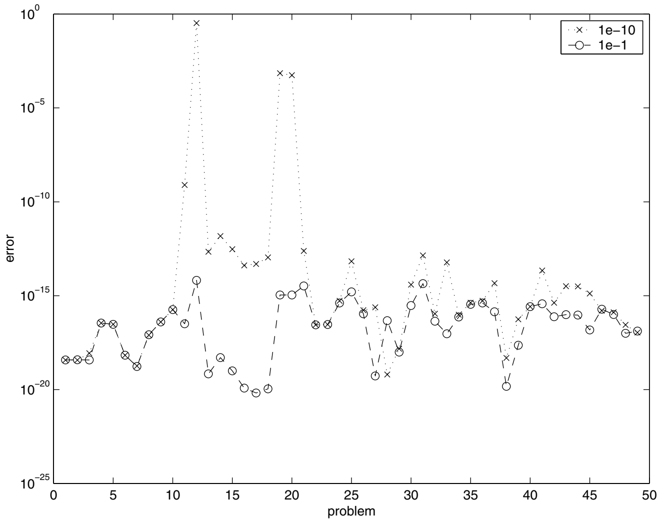


Fig. 6. The relative residual for the dynamic LDL^T factorization, using the multifrontal algorithm, for pivoting thresholds of 1e-1 and 1e-10, respectively

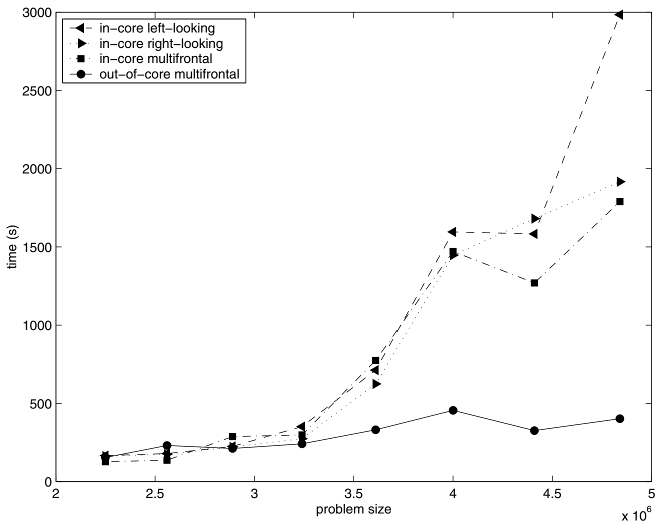


Fig. 7. The execution time of the $\tilde{L}\tilde{L}^T$ factorization, in-core (left-looking, right-looking and multifrontal) and out-of-core (multifrontal) (Sun UltraSparc III platform: 900 MHz, 2GB). There is only a slight increase of the out-of-core execution time because a slight increase of the problem size determines a slight increase of the amount of explicit I/O. The non-monotonicity of the plots is caused by the lack of smoothness in the ordering algorithm

elegant, is to use run time type identification. This is the solution that we adopted in Oblio. Oblio identifies the data structure at run time and performs I/O only in the out-of-core context. The identification of the actual data structure is done at the highest level and thus virtually comes at no cost.

In Oblio, most of the basic arithmetic operations are efficiently implemented through level 3 BLAS/LAPACK calls.

3 Performance

We present a reduced set of results here. More details will be available in a future paper. We also refer the reader to [12] for a recent comparative study of sparse symmetric direct solvers (including Oblio). In all the experiments below we use the node nested dissection algorithm from METIS [6] as a fill reducing ordering.

As reported in [12] Oblio performs as well as other similar solvers for positive definite problems and correctly solves indefinite problems. For the latter Oblio is not as fast as other indefinite solvers since we have not yet optimized our indefinite kernels. The best factorization rate observed so far is 42% of the peak rate on an IBM Power 4 platform (1.3 GHz, 5200 Mflop/s peak rate).

In-Core Results. Figures 2 through 6 illustrate in-core experiments performed with Oblio. All problems are indefinite and represent a subset of the collection used in [12]. We used dynamic LDL^T factorization and, for comparison, we used $\tilde{L}\tilde{L}$ factorization as well, after replacing the original numerical entries and transforming indefinite matrices into positive definite matrices. We used the in-core dynamic factor data structure for the former and the in-core static factor data structure for the latter, and we employed all three factorization algorithms. The results are obtained on an IBM Power 3 platform (375 MHz, 16 GB), which we chose because we were particularly interested in a large amount of internal memory.

Figure 2 shows the difference in storage requirement between the three algorithms, in the positive definite case. The plot represents the ratio between the storage required by the multifrontal algorithm and the storage required by the left-looking and right-looking algorithms. This indicates how much additional storage is required by the multifrontal algorithm. For a significant number of problems the amount of additional storage is not large and the multifrontal algorithm is expected to perform well.

Execution time results for positive definite problems are shown in Fig. 3. The plots represent the ratios between the times required to factor using the left-looking and multifrontal algorithms, and between the times required to factor using the right-looking and multifrontal algorithms, respectively. Note that the multifrontal algorithm outperforms the other two for many problems.

Figures 4 and 5 illustrate results for the indefinite case, using only the multifrontal algorithm and two different pivoting thresholds, 1e-1 and 1e-10. The plots show the increase in storage and execution time (same IBM Power 3 platform). The increase is smaller when the pivoting threshold is smaller but a larger pivoting threshold may be required for accurate results. Similar results can be obtained with the indefinite left-looking and right-looking algorithms.

For indefinite problems we also report accuracy results. These are provided in Fig. 6. The plots represent the relative residual for the two sets of experiments. For most of the problems from this set the computation is accurate enough with the relaxed pivoting threshold (1e-10) but some problems require a larger pivoting threshold, at the cost of a more expensive computation.

Out-of-Core Results. For the out-of-core experiments we chose a platform with a smaller amount of internal memory: an UltraSparc III Sun Fire 280R (900 MHz, 2GB). Since this platform has a smaller core we can easily determine differences in performance between the in-core and the out-of-core computations.

Unfortunately, matrix collections usually provide problems of a given size, even if some of these problems may be large. In order to run out-of-core experiments one should be able to tune the problem size given a particular core size. Here, in order to match the 2 GB core size we tune the order of the problem from roughly 2,000,000 to roughly 5,000,000 (this way we cross the 2GB threshold). We use model 2d finite difference discretization grids with 5-point stencils, the grid size ranging from 1,500 to 2,200.

Figure 7 plots the execution time for the three in-core $\tilde{L}\tilde{L}^T$ factorizations (static data structure) as well as the execution time for the out-of-core multifrontal $\tilde{L}\tilde{L}^T$ factorization. As expected, the in-core factorizations start to perform poorly close to the 2GB threshold. At that point the operating system begins to rely on virtual memory in order to provide the required storage. This translates into *implicit I/O* (swapping), which slows down the computation. On the other hand, the out-of-core factorization performs *explicit I/O* and therefore the computation continues to be performed efficiently.

4 Conclusion

Oblio is an active project and the package continues to be developed. Our current effort is targeted toward software design improvements and code optimizations. We also plan several extensions of the package, mostly in order to provide more options for numerically difficult problems. A future, longer paper will address the design of Oblio in more detail and will provide more results.

References

1. P. R. Amestoy, I. S. Duff, J. Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.
2. C. Ashcraft, R. G. Grimes, and J. G. Lewis. Accurate symmetric indefinite linear equation solvers. *SIAM Journal on Matrix Analysis and Applications*, 20(2):513–561, 1998.
3. F. Dobrian, G. K. Kumfert, and A. Pothen. The design of sparse direct solvers using object-oriented techniques. In H. P. Langtangen, A. M. Bruaset, and E. Quak, editors, *Advances in Software Tools in Scientific Computing*, volume 50 of *Lecture Notes in Computational Science and Engineering*, pages 89–131. Springer-Verlag, 2000.
4. I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22(4):973–996, 2001.
5. A. Gupta, M. Joshi, and V. Kumar. WSMP: A high-performance shared- and distributed-memory parallel sparse linear equation solver. Technical Report RC 22038, IBM T.J. Watson Research Center, 2001.

6. G. Karypis and V. Kumar. METIS: Family of multilevel partitioning algorithms. http://www-users.cs.umn.edu/˜karypis/metis.
7. X. S. Li and J. W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29(2):110–140, 2003.
8. A. Pothen and S. Toledo. Elimination structures in scientific computing. In D. Mehta and S. Sahni, editors, *Handbook on Data Structures and Applications*, pages 59.1–59.29. CRC Press, 2004.
9. E. Rothberg and R. Schreiber. Efficient methods for out-of-core sparse Cholesky factorization. *SIAM Journal on Scientific Computing*, 21(1):129–144, 1999.
10. V. Rotkin and S. Toledo. The design and implementation of a new out-of-core sparse Cholesky factorization method. *ACM Transactions on Mathematical Software*, 30(1):19–46, 2004.
11. O. Schenk and K. Gärtner. Solving unsymmetric sparse systems of linear equations with PARDISO. *Future Generation Computer Systems*, 20(3):475–487, 2004.
12. J. A. Scott, Y. Hu, and N. I. M Gould. An evaluation of sparse direct symmetric solvers: an introduction and preliminary findings. Numerical Analysis Internal Report 2004-1, Rutherford Appleton Laboratory, 2004.

Performance Analysis of Parallel Right-Looking Sparse LU Factorization on Two Dimensional Grids of Processors

Laura Grigori[1] and Xiaoye S. Li[2]

[1] INRIA Rennes
Campus Universitaire de Beaulieu, 35042 Rennes, France
Laura.Grigori@irisa.fr

[2] Lawrence Berkeley National Laboratory, MS 50F-1650
One Cyclotron Road, Berkeley, CA 94720, USA
xsli@lbl.gov

Abstract. We investigate performance characteristics for the LU factorization of large matrices with various sparsity patterns. We consider supernodal *right-looking* parallel factorization on a two dimensional grid of processors, making use of static pivoting. We develop a performance model and we validate it using the implementation in SuperLU_DIST, the real matrices and the IBM Power3 machine at NERSC. We use this model to obtain performance bounds on parallel computers, to perform scalability analysis and to identify performance bottlenecks. We also discuss the role of load balance and data distribution in this approach.

1 Introduction

A valuable tool in designing a parallel algorithm is to analyze its performance characteristics for various classes of applications and machine configurations. Very often, good performance models reveal communication inefficiency and memory access contention that limit the overall performance. Modeling these aspects in detail can give insights into the performance bottlenecks and help improve the algorithm. The goal of this paper is to analyze performance characteristics and scalability for the LU factorization of large matrices with various sparsity patterns.

For dense matrices, the factorization algorithms have been shown to exhibit good scalability, where the efficiency can be approximately maintained as the number of processors increases when the memory requirements per processor are held constant [2]. For sparse matrices, however, the efficiency is much harder to predict since the sparsity patterns vary with different applications. Several results exist in the literature [1,4,6], which were obtained for particular classes of matrices arising from the discretization of a physical domain. They show that factorization is not always scalable with respect to memory use. For sparse matrices resulting from two-dimensional domains, the best parallel algorithm lead to an increase of the memory at a rate of $O(P \log P)$ with increasing P [4]. It is worth mentioning that for matrices resulting from three-dimensional domains, the best algorithm is scalable with respect to memory size.

In this work, we develop a performance model for a sparse factorization algorithm that is suitable for analyzing performance with arbitrary input matrix. We use a classical model to describe an ideal machine architecture in terms of processor speed, network latency and bandwidth. Using this machine model and several input characteristics (order of the matrix, number of nonzeros, etc), we analyze a supernodal *right-looking* parallel factorization on two dimensional grids of processors, making use of static pivoting. This analysis allows us to obtain performance upper bounds on parallel computers, to perform scalability analysis, to identify performance bottlenecks and to discuss the role of load balance and data distribution. More importantly, our performance model reveals the relationship between parallel runtime and matrix sparsity, where the sparsity is measured with respect to the underlying hardware's characteristics. Given any combination of application and architecture, we can obtain this sparsity measure. Then our model can quantitatively predict not only the performance on this machine, but also what hardware parameters to improve are most critical to improve the performance for this type of applications. We validate our analytical model using the actual factorization algorithm implemented in the SuperLU_DIST [5] solver, the real-world matrices and the IBM Power3 machine at NERSC. We also show that the runtime predicted by our model is more accurate than that predicted by simply examining the workload on the critical path, because our model takes into account both task dependency and communication overhead.

The rest of the paper is organized as follows: Section 2 introduces a performance analysis model for the right-looking factorization, with its scalability analysis. The experimental results validating the performance model are presented in Section 3 and Section 4 draws the conclusions.

2 Parallel Right-Looking Sparse LU Factorization on Two Dimensional Grids of Processors

Consider factorizing a sparse unsymmetric $n \times n$ matrix A into the product of a unit lower triangular matrix L and an upper triangular matrix U. We discuss a parallel execution of this factorization on a two dimensional grid of processors. The matrix is partitioned into $N \times N$ blocks of submatrices using unsymmetric supernodes (columns of L with the same nonzero structure). These blocks of submatrices are further distributed among a two dimensional grid $P_r \times P_c$ of P processors ($P_r \times P_c \leq P$) using a block cyclic distribution. With this distribution, a block at position (I, J) of the matrix ($0 \leq I, J < N$) will be mapped on the process at position $(I\, mod P_r, J\, mod P_c)$ of the grid. $U(K, J)$ ($L(K, J)$) denotes a submatrix of U (L) at row block index K and column block index J.

The algorithm below describes a right-looking factorization and Figure 1 illustrates the respective execution on a rectangular grid of processors. This algorithm loops over the N supernodes. In the K-th iteration, the first $K-1$ block columns of L and block rows of U are already computed. At this iteration, first the column of processors owning block column K of L factors this block column $L(K : N, K)$; second, the row of processors owning block row K of U performs the triangular solve to compute $U(K, K+1 : N)$; and third, all the processors update the trailing matrix using $L(K+1 : N, K)$ and $U(K, K+1 : N)$. This third step requires most of the work and also exhibits most of the parallelism in the right-looking approach.

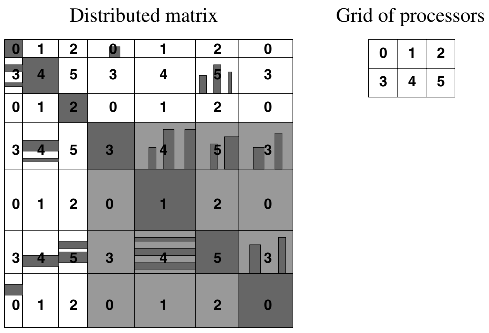


Fig. 1. Illustration of parallel right-looking factorization

```
for $K := 1$ to $N$ do
  Factorize block column $L(K : N, K)$
  Perform triangular solves: $U(K, K+1 : N) := L(K,K)^{-1} \times A(K, K+1 : N)$
  for $J := K+1$ to $N$ with $U(K,J) \neq 0$ do
    for $I := K+1$ to $N$ with $L(I,K) \neq 0$ do
      Update trailing submatrix:
      $A(I,J) := A(I,J) - L(I,K) \times U(K,J)$
    end for
  end for
end for
```

The performance model we develop for the sparse LU factorization is close to the performance model developed for the dense factorization algorithms in ScaLAPACK [2]. Processors have local memory and are connected by a network that provides each processor direct links with any of its 4 direct neighbors (mesh-like).

To simplify analysis and to make the model easier to understand, we make the following assumptions:

- We use one parameter to describe the processor flop rate, denoted γ, and we ignore communication collisions. We estimate the time for sending a message of m items between two processors as $\alpha + m\beta$, where α denotes the latency and β the inverse of the bandwidth.
- We approximate the cost of a broadcast to p processors by $\log p$ [2]. Furthermore, the LU factorization uses a pipelined execution to overlap some of the communication with computation, and in this case the cost of a broadcast is estimated by 2 [2].
- We assume that the computation of each supernode lies on the critical path of execution, that is the length of the critical path is N. We also assume that the load and the data is evenly distributed among processors. Later in Section 3, we will provide the experimental data verifying these assumptions.

Runtime Estimation. We use the following notations to estimate the runtime to factorize an $n \times n$ matrix. We use c_k to denote the number of off-diagonal elements in each column of block column K of L, r_k to denote the number of off-diagonal elements in each row of block row K of U, $nnz(L)$ to denote the number of nonzeros in the off-diagonal blocks of L, $nnz(U)$ to denote the number of nonzeros in the off-diagonal blocks of U.

$M = 2\sum_{k=1}^{n} c_k r_k$ is the total number of flops in the trailing matrix update, counting both multiplications and additions. $F = nnz(L) + M$ is the total number of flops in the factorization.

With the above notations, the sequential runtime can be estimated as

$$T_s = nnz(L)\gamma + M\gamma = F\gamma.$$

We assume each processor in the column processors owning block column K gets $s \cdot c_k/P_r$ elements, where $s \cdot c_k$ is the number of nonzeros in the block column K and P_r is the number of processors in the column. Block row K of U is distributed in a similar manner.The parallel runtime using a square grid of processors can be expressed as:

$$T(N, \sqrt{P} \times \sqrt{P}) \approx \frac{F}{P}\gamma + (2N + \frac{1}{2}N\log P)\alpha + \frac{(2nnz(L) + \frac{1}{2}nnz(U)\log P)}{\sqrt{P}}\beta$$

The first term represents the parallelization of the computation. The second term represents the number of broadcasting messages. The third term represents the volume of communication overhead.

Scalability Analysis. We now examine the scalability using a square grid of processors of size P, where the efficiency of the right-looking algorithm is given by the following formula:

$$\begin{aligned} \epsilon(N, \sqrt{P} \times \sqrt{P}) &= \frac{T_s(N)}{PT(N, \sqrt{P} \times \sqrt{P})} && (2.1) \\ &\approx \left[1 + \frac{NP\log P}{F}\frac{\alpha}{\gamma} + \frac{(2nnz(L) + nnz(U)\log P)\sqrt{P}}{F}\frac{\beta}{\gamma}\right]^{-1} && (2.2) \end{aligned}$$

One interesting question is which of the three terms dominates efficiency (depending on the size and the sparsity of the matrix). The preliminary remark is that, for very dense problems (F large), the first term significantly affects the parallel efficiency.

For the other cases, we can compare the last two terms to determine which one is dominant. That is, if we ignore the factors 2 and $\log P$ in the third term, we need to compare $\sqrt{P}\frac{\alpha}{\beta}$ with $\frac{nnz(L+U)}{N}$. Assuming that the network's latency-bandwidth product is given ($\frac{\alpha}{\beta}$), we can determine if the ratio of the latency to the flop rate (α/γ term) or the ratio of the inverse of the bandwidth to the flop rate (β/γ term) dominates efficiency. Overall, the following observations hold:

Case 1 For sparser problems ($\sqrt{P}\frac{\alpha}{\beta} > \frac{nnz(L+U)}{N}$), the α/γ term dominates efficiency.

Case 2 For denser problems ($\sqrt{P}\frac{\alpha}{\beta} < \frac{nnz(L+U)}{N}$), the β/γ term dominates efficiency.

Case 3 For problems for which $\frac{nnz(L+U)}{N}$ is close to $\sqrt{P}\frac{\alpha}{\beta}$, the β/γ term can be dominant on smaller number of processors, and with increasing number of processors the α/γ term can become dominant.

Note that even for Case 2, the algorithm behaviour varies during the factorization: at the beginning of the factorization, where the matrix is generally sparser and the messages are shorter, α/γ term dominates the efficiency, while at the end of the factorization where the matrix becomes denser, β/γ term dominates the efficiency.

Let us now consider matrices in Case 2. For these matrices, in order to maintain a constant efficiency, $\frac{F}{nnz(L+U)}$ must grow proportionally with $\sqrt{P}$. On the other hand, for a scalable algorithm in terms of memory requirements, the memory requirement $nnz(L+U)$ should not grow faster than P. Thus, when we allow $nnz(L+U) \propto P$, the condition $F \propto nnz(L+U)^{3/2}$ must be satisfied, and the efficiency can be approximately maintained constant. (In reality, even for these matrices, α/γ term as well as $\log P$ factor will still contribute to efficiency degradation.) We note that the matrices with N unknowns arising from discretization of Laplacian operator on three-dimensional finite element grids fall into this category. Using nested dissection, the number of fill-ins in such matrix is on the order of $O(N^{4/3})$ while the amount of work is on the order of $O(N^2)$. Maintaining a fixed efficiency requires that the number of processors P grows proportionally with $N^{4/3}$, the size of the factored matrix. In essence, the efficiency for these problems can be approximately maintained if the memory requirement per processor is constant. Note that α/γ term grows with $N^{1/3}$, which also contributes to efficiency loss.

3 Experimental Results

In this section, we compare the analytical results against experimental results obtained on a IBM Power3 machine at NERSC, with real-world matrices. The test matrices and their characteristics are presented in Table 1. They are ordered according to the last column, $nnz(L+U)/N$, which we use as a measure of the sparsity of the matrix.

Table 1. Benchmark matrices

Matrix	Order	N	$nnz(A)$	$nnz(L+U)$	$Flops(F)$	$nnz(L+U)/N$
				$\times 10^6$	$\times 10^9$	$\times 10^3$
af23560	23560	10440	484256	11.8	5.41	1.13
rma10	46835	6427	2374001	9.36	1.61	1.45
ecl32	51993	25827	380415	41.4	60.45	1.60
bbmat	38744	11212	1771722	35.0	25.24	3.12
inv-extr1	30412	6987	1793881	28.1	27.26	4.02
ex11	16614	2033	1096948	11.5	5.60	5.65
fidapm11	22294	3873	623554	26.5	26.80	6.84
mixingtank	29957	4609	1995041	44.7	79.57	9.69

The first goal of our experiments is to analyze the different assumptions we have made during the development of the performance model. Consider again the efficiency Equation (2.2) in Section 2. For the first term, we assume that the load is evenly distributed among processors, while for the third term we assume that the data is evenly distributed among processors. Note that we also assume that the computation of each supernode lies on the critical path of execution. Our experiments show that a two dimensional distribution of the data on a two dimensional grid of processors leads to a balanced distribution of the data. They also show that for almost all the matrices, the critical path assumption is realistic [3].

However, the load balance assumption is not always valid. To assess load balance, we consider the load F to be the number of floating point operations to factorize the matrix. We then compute the load lying on the critical path F_{CP} by adding at each iteration the load of the most loaded processor in this iteration. More precisely, consider f_{pi} being the load of processor p at iteration i (number of flops performed by this processor at iteration i). Then $F_{CP} = \sum_{i=1}^{N} \max_{p=1}^{P} f_{pi}$. The load balance factor is computed as $LB = \frac{F_{CP}P}{F}$. In other words, LB is the load of heaviest processors lying on the critical path divided by the average load per processor. The closer this factor approaches 1, the better is the load balance. Contrary to the "usual" way, when the load balance factor is computed as the average load divided by the maximum load among all the processors, our computation of load balance is more precise. Note that we can also use $\frac{F}{F_{CP}}$ to compute a crude upper bound on the parallel speedup, which takes into account the workload on the critical path but ignores communication cost and task dependency. The results are presented in Table 2, in the rows denoted by LB. We observe that the workload distribution is good for large matrices on a small number of processors. But it can degrade quickly for some matrices such as rma10, for which load balance can degrade by a factor of 2 when increasing the numbers of processors by a factor of 2. Consequently, efficiency will suffer a significant degradation.

The second goal of the experiments is to show how the experimental results support our analytical performance model developed in Section 2. For this, we use the analytical performance model to predict the speedup that each matrix should attain with an increasing number of processors. Then we compare the predicted speedup against the speedup obtained by SuperLU_DIST. The plots in Figure 2 display these results, where the predicted speedup for each matrix M is denoted by Mp, and the actually obtained (measured) speedup is denoted by Mm. We also display in these plots the upper bound obtained from the workload on the critical path, given by $\frac{F}{F_{CP}}$, and we denote it as MLB.

As the plots show, the analytical performance model predicts well the performance on a small number of processors (up to 30-40 processors), while the predicted speedup starts to deviate above the measured speedup with an increase in the number of processors. This is because on a smaller number of processors our model assumptions are rather realistic, but on a larger number of processors the assumptions are deviating from reality. That is why we see the degraded scalability. More detailed data were reported in [3].

The upper bound based only on the workload on the critical path can be loose, since it ignores communication and task dependency. However, it often corroborates the general trend of the speedup predicted by our analytical model. For several matrices, such as bbmat, inv-extr1, fidapm11 and mixingtank, this upper bound is very close to the prediction given by our analytical model (see Figure 2), implying that for those matrices, load imbalance is a more severe problem than communication, and improving load balance alone can greatly improve the overall performance.

Table 2. Runtimes (in seconds) and load distribution (LB) for right-looking factorization on two dimensional grids of processors

		P=1	P = 4	P = 16	P = 32	P = 64	P = 128
af23560	time	9.95	3.95	2.36	2.30	3.17	3.38
	LB	1.0	1.28	2.05	2.92	4.21	6.67
rma10	time	3.41	2.12	1.90	1.99	3.03	3.17
	LB	1.0	1.66	2.75	5.62	9.13	16.07
ecl32	time	104.27	29.34	9.49	7.25	7.31	7.22
	LB	1.0	1.09	1.28	1.52	1.80	2.37
bbmat	time	67.37	19.50	7.61	5.64	6.05	6.50
	LB	1.0	1.21	1.75	2.35	3.17	4.88
inv-extr1	time	73.13	19.08	6.46	4.72	4.95	5.29
	LB	1.0	1.14	1.42	1.87	2.45	3.51
ex11	time	9.49	3.27	1.54	1.33	1.65	2.07
	LB	1.0	1.27	1.90	2.58	3.53	5.32
fidapm11	time	51.99	14.21	4.84	3.57	3.47	3.81
	LB	1.0	1.15	1.46	1.83	2.31	3.13
mixingtank	time	119.45	33.87	9.52	6.47	5.53	5.27
	LB	1.0	1.08	1.25	1.43	1.63	2.04

The third goal of the experiments is to study the actual efficiency case by case for all the test matrices. One approach to do this is to observe how the factorization runtime degrades as the number of processors increases for different matrices. For this we report in Table 2 the runtime in seconds of SuperLU_DIST factorization. These results illustrate that good speedups can be obtained on a small number of processors and show how efficiency degrades on a larger number of processors. As one would expect, the efficiency degrades faster for problems of smaller size (number of flops smaller), and slower for larger problems.

We now examine how the actual model parameters (sparsity, α, β and γ) affect the performance. On the IBM Power3, the measured latency is 8.0 microseconds and the bandwidth ($1/\beta$) for our medium size of messages is 494 MB/s [7]. The latency-bandwidth product is $\alpha/\beta = 4 \times 10^3$. Table 1 shows that the algorithm's efficiency for some matrices is clearly dominated by the α/γ term, such as af23560, rma10, ecl32 (Case 1 matrices). For the other matrices, mixingtank, fidapm11, ex11, inv-extr1, the efficiency is significantly affected by the β/γ term (Case 2 matrices).

Matrices af23560 and ex11 have an approximately equal number of flops, and almost similar runtimes on one processor. But efficiency degrades faster for af23560 than for ex11. This is because the efficiency of af23560 is mostly affected by the α/γ term (Case 1), while the efficiency of ex11 is mainly affected by the β/γ term (Case 2), and thus its performance degrades slower than for af23560. For denser matrices which fall into Case 2, such as mixingtank, the algorithm achieves much better efficiency even on

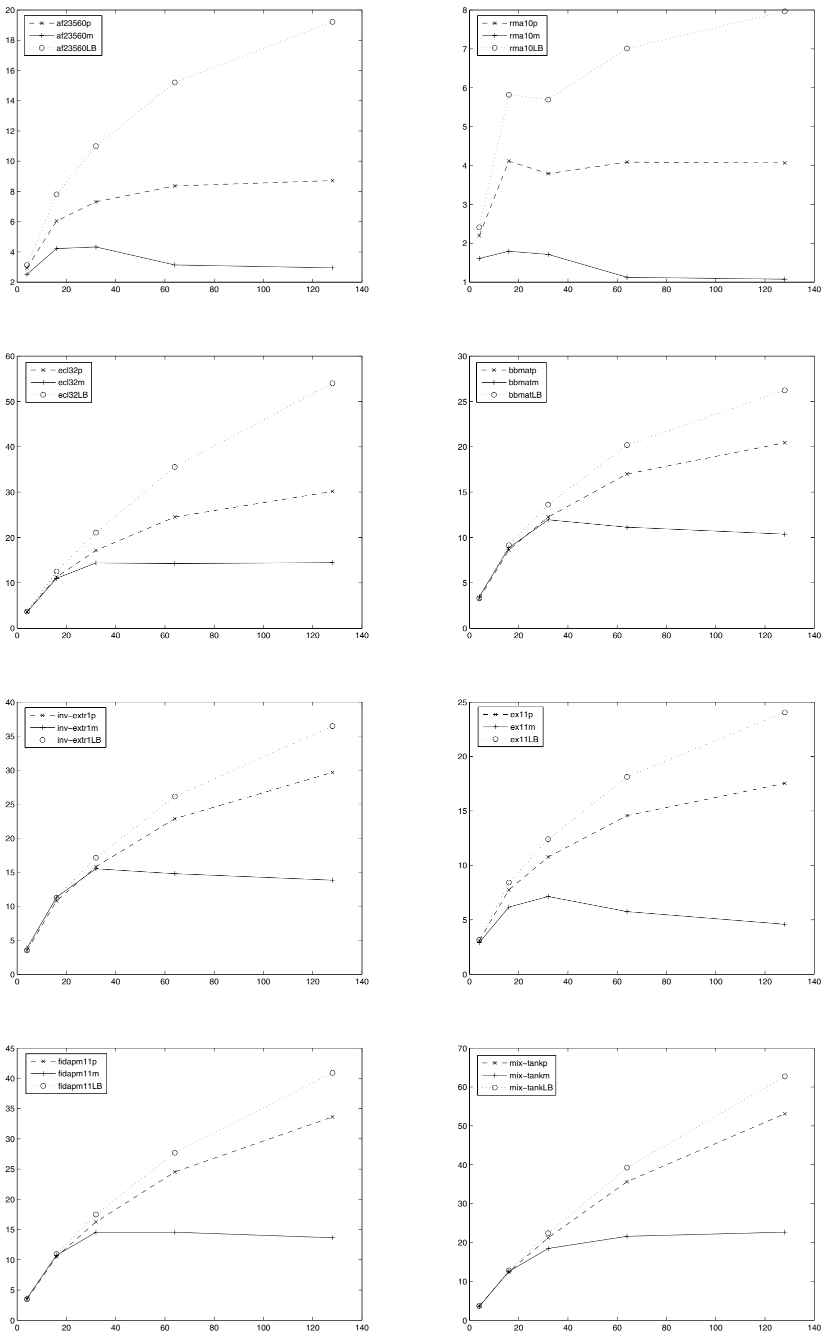


Fig. 2. Speedups predicted by our performance model (labeled “p”) and by the load balance constraint (labeled “LB”), versus the measured speedups (labeled “m”)

large number of processors. Therefore, the algorithm is more sensitive to latency than bandwidth.

4 Conclusions

We developed a performance model for a sparse right-looking LU factorization algorithm and validated this model using the SuperLU_DIST solver, real-world matrices and the IBM Power3 machine at NERSC.

Using this model, we first analyzed the efficiency of this algorithm with increasing number of processors and problem size. We concluded that for matrices satisfying a certain relation (namely $F \propto nnz(L+U)^{3/2}$) between their problem size and their memory requirements, the algorithm is scalable with respect to memory use. This relation is satisfied by matrices arising from the 3D model problems. For these matrices the efficiency can be roughly maintained constant when the number of processors increases and the memory requirement per processor is constant. But for matrices arising from the 2D model problems, the algorithm is not scalable with respect to memory use, the same as sparse Cholesky factorization [4].

Secondly, we analyzed the efficiency of this algorithm for fixed problem size and increasing number of processors. We observed that good speedups can be obtained on smaller number of processors. On larger number of processors, the efficiency degrades faster for sparser problems which are more sensitive to the latency of the network. A two dimensional distribution of the data on a two dimensional grid of processors leads to a balanced distribution of the data. It also leads to a balanced distribution of the load on smaller number of processors. But the load balance is usually poor on larger number of processors. We believe that load imbalance and insufficient amount of work (F) relative to communication overhead are the main sources of worse efficiency on large number of processors.

One practical use of our theoretical efficiency bound is as follows. For certain application domain, the matrices usually exhibit a similar sparsity pattern. We can measure the sparsity with respect to the underlying machine parameters, i.e., floating-point speed, the network latency and bandwidth. Depending on whether they belong to Case 1 or Case 2, we can determine the most critical hardware parameters which need to be improved in order to enhance the performance for this class of application. In addition, given several choices of machines, we can predict which hardware combination is best for this application.

References

1. Cleve Ashcraft. The fan-both family of column-based distributed Cholesky factorization algorithms. In Alan George, John R. Gilbert, and Joseph W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, pages 159–191. Springer Verlag, 1994.
2. Jack K. Dongarra, Robert A. van de Geijn, and David W. Walker. Scalability Issues Affecting the Design of a Dense Linear Algebra Library. *Journal of Parallel and Distributed Computing*, 22(3):523–537, 1994.
3. Laura Grigori and Xiaoye S. Li. Performance analysis of parallel supernodal sparse lu factorization. Technical Report LBNL-54497, Lawrence Berkeley National Laboratory, Feburary 2004.

4. Anshul Gupta, George Karypis, and Vipin Kumar. Highly Scalable Parallel Algorithms for Sparse Matrix Factorization. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):502–520, 1997.
5. Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110–140, June 2003.
6. Robert Schreiber. Scalability of sparse direct solvers. In Alan George, John R. Gilbert, and Joseph W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, pages 191–211. Springer Verlag, 1994.
7. Adrian Wong. Private communication. Lawrence Berkeley National Laboratory, 2002.

A Shared- and Distributed-Memory Parallel Sparse Direct Solver

Anshul Gupta

IBM T.J. Watson Research Center
1101 Kitchawan Road
Yorktown Heights, NY 10598, USA

1 Introduction

In this paper, we describe a parallel direct solver for general sparse systems of linear equations that has recently been included in the Watson Sparse Matrix Package (WSMP) [7]. This solver utilizes both shared- and distributed- memory parallelism in the same program and is designed for a hierarchical parallel computer with network-interconnected SMP nodes. We compare the WSMP solver with two similar well known solvers: MUMPS [2] and Super_LU$_{Dist}$ [10]. We show that the WSMP solver achieves significantly better performance than both these solvers based on traditional algorithms and is more numerically robust than Super_LU$_{Dist}$. We had earlier shown [8] that MUMPS and Super_LU$_{Dist}$ are amongst the fastest distributed-memory general sparse solvers available.

The parallel sparse LU factorization algorithm in WSMP is based on the unsymmetric pattern multifrontal method [3]. The task- and data-dependency graph for symmetric multifrontal factorization is a tree. A task-dependency graph (henceforth, task-DAG) is a directed acyclic graph in which the vertices correspond to the tasks of factoring rows or columns or groups of rows and columns of the sparse matrix and the edges correspond to the dependencies between the tasks. A task is ready for execution if and only if all tasks with incoming edges to it have completed. The vertex set of a data-dependency graph (henceforth, data-DAG) is the same as that of the task-DAG and and edge from vertex i to a vertex j denotes that at least some of the output data of task i is required as input by task j. In unsymmetric pattern multifrontal factorization, the task- and data-DAGs are general directed acyclic graphs in which nodes can have multiple parents.

Hadfield [9] introduced a parallel sparse LU factorization code based on the unsymmetric pattern multifrontal method. A significant drawback of this implementation was that partial pivoting during factorization would change the row/column order of the sparse matrix. Therefore, the data-DAG needed to be generated during numerical factorization, thus introducing considerable symbolic processing overhead. We recently introduced improved symbolic and numerical factorization algorithms for general sparse matrices [6]. The symbolic algorithms are capable of inexpensively computing a minimal task-dependency graph and near-minimal data-dependency graph for factoring a general sparse matrix. These graphs, computed solely from the nonzero pattern of the sparse matrix, are valid for any amount of pivoting induced by the numerical values during LU factorization. With the introduction of these algorithms, the symbolic processing phase can be completely separated from numerical factorization and needs to be performed only once for matrices with the same initial structure but different numerical values, and hence, potentially different pivoting sequences during numerical factorization. A description of the serial unsymmetric pattern multifrontal algorithm used in WSMP can be found in [6].

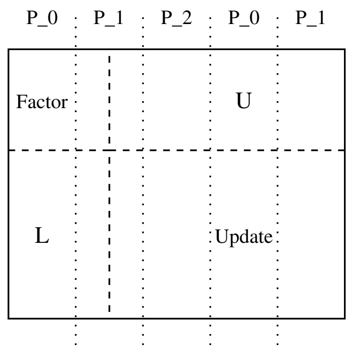
2 Overview of the Parallel Sparse LU Algorithm

WSMP is designed to make the best use of the computational resources of most modern parallel computers. These machines, typically, are either shared-memory multiprocessors or are clusters of nodes consisting of shared-memory multiprocessors. WSMP can run on multiple nodes using MPI processes and each process uses threads to utilize all the CPUs on the node.

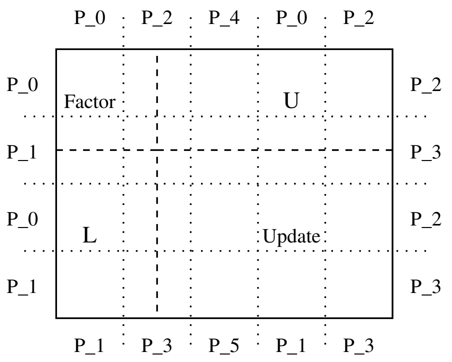
The symbolic preprocessing step, among other things, generates a static data-DAG that defines the communication and computation pattern of the multifrontal LU factorization. This static information is used to generate a static mapping of the data-DAG onto the processes. The work required to processes certain nodes could change during execution due to pivoting. However, such changes are usually relatively small and are randomly distributed among the processes. Therefore, they rarely pose a serious load-imbalance problem. Dynamic load-balancing would have introduced extra communication overhead and other complexities in the code. With these factors in mind, we chose static load-balancing at the process level. However, multiple threads running on each process keep their loads balanced dynamically. Each process maintains a dynamic heap of tasks that are ready for execution. The threads dynamically pick tasks from the heap and add more tasks as they become available. Further discussion of the SMP-parallel component of the factorization can be found in [5].

With the exception of the root node, a typical task in WSMP's multifrontal factorization involves partial factorization and update of a dense rectangular frontal matrix and distributing the updated part of the matrix among the parents of the node corresponding to the factored frontal matrix. The root node involves full factorization of a dense square matrix. Depending on its size and location relative to the root, a task may be mapped onto one or multiple nodes. When mapped on a single node, a task may be performed by a single or by multiple threads. If required, appropriate shared-memory parallel dense linear algebra routines are employed for partial factorization and updates.

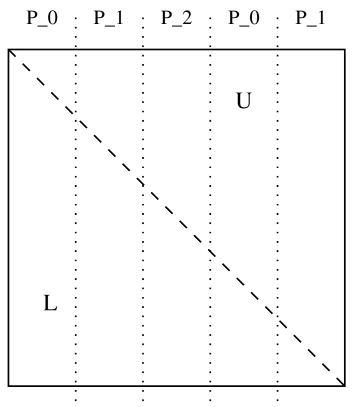
When a task is mapped onto multiple processes, the group of processes on which the task is mapped is viewed as a virtual grid. The virtual grid can be one-dimensional or two-dimensional, depending on the number of processes. In the current implementation, the grids are chosen such that they are one-dimensional with multiple columns for less than 6 processes and are two-dimensional for 6 or more processes. Moreover, the number of process rows in all grids is always a power of 2. All grid sizes are not allowed. For example, a 5-process grid is 1×5, a 6-process grid is 2×3, a 7-process grid is not allowed, and an 8-process grid is 2×4. The root node is always mapped onto the largest permissible grid with number of processes less than or equal to the total number of MPI processes that the program is running on. As we move away from the root node in the data-DAG and as more tasks can be executed in parallel, the process grids onto which



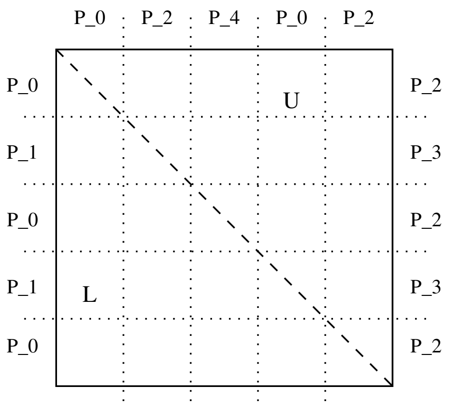
(a) A partial factor and update task mapped onto a 3−process 1−dimensional grid.



(b) A partial factor and update task mapped onto a 6−process 2−dimensional grid.



(c) A factorization task corresponding to the root node mapped onto a 3−process 1−dimensional grid.



(d) A factorization task corresponding to the root node mapped onto a 6−process 2−dimensional grid.

Fig. 1. The four scenarios for mapping frontal matrices onto process grids

these tasks are mapped become progressively smaller. Eventually, the tasks are mapped onto single processes.

In addition to a serial/multithreaded partial factorization kernel, four types of message-passing parallel factorization kernels are employed for the four scenarios for mapping frontal matrices onto process grid, as shown in Figure 1. Efficient implementation of pivoting for numerical stability is a key requirement of these kernels.

With a non-root frontal matrix mapped onto a one-dimensional grid (Figure 1(a)), if a process can find a pivot amongs its columns, then no communication is required for pivoting. Otherwise, a column interchange involving communication with another process is required. When a non-root frontal matrix is mapped onto a two-dimensional grid (Figure 1(b)), then finding a pivot may require all processes in a column of the grid to communicate to find the pivot. That is the reason why the process grids have fewer rows than columns and the number of rows is a power of two so that fast logarithmic time communication patterns can be used for pivot searches along columns. Furthermore, the WSMP algorithm avoids this communication at each pivot step by exchanging data corresponding to more than one column in each communication step. The frontal matrix corresponding to the root supernode never requires column interchanges because this matrix is factored completely. Therefore, pivoting is always free of communication for this matrix on a one-dimensional grid (Figure 1(c)). On a two-dimensional grid (Figure 1(d)), pivoting on the root supernode involves communication among the processes along process columns.

Once a pivot block has been factored, it is communicated along the pivot row and the pivot column, which are updated and are then communicated along the columns and the rows of the grid, respectively, to update the remainder of the matrix. The computation then moves to the next pivot block in a pipelined manner and continues until the supernode has been factored completely or no more pivots can be found. Finally, the update matrix is distributed among the parent supernodes for inclusion into their frontal matrices.

3 Performance Comparison

In this section, we compare the performance of WSMP with that of MUMPS [2] (version 4.3.2) and Super_LU$_{Dist}$ [10] (version 2.0) on a suite of 24 test matrices on one to eight 1.45 GHz Power4+ CPUs of an IBM p650 SMP computer. All codes were compiled in 32-bit mode and each process had access to 2 GB of memory. The *MP_SHARED_MEMORY* environment variable was set to *yes* to enable MPI to take advantage of shared memory for passing messages. All codes were used with their default options, including their default fill-reducing ordering. WSMP uses its own nested-dissection based ordering, MUMPS uses a nested-dissection based ordering software called PORD [11], and Super_LU$_{Dist}$ uses the approximate minimum degree algorithm [1]. All three softwares use row prepermutation to permute large entries of the matrix to the diagonal [4]. A detailed description of the various algorithms and features of these packages can be found in [8].

Figures 2 and 3 show a comparison of the times taken by WSMP and MUMPS to solve a single system of equation on 1, 2, 4, and 8 CPUs with medium to large general sparse coefficient matrices. In each case, the total solution time of WSMP is considered to be one unit and all other times are normalized with respect to this. Each bar is further divided into a solid and an empty portion depicting the portions of the total time spent in factorization and triangular solves. The solution phase depicted by the hollow portion of the bars includes the iterative refinement time to reduce the residual of the backward error to 10^{-15}. Some bars are missing for some of the largest matrices because the program could not complete due to insufficient memory. Figure 4 shows a similar comparison between WSMP and Super_LU$_{Dist}$, but contains bars corresponding to the later only. Since MUMPS and Super_LU$_{Dist}$ are designed to work only in distributed-memory parallel mode, all three codes are run with a single thread on each process for this comparison.

Figures 2 and 3 reveal that on a single CPU, MUMPS typically takes between 1 and 2 times the time that WSMP takes to solve a system of equations. However, WSMP appears to be more scalable. With very few exceptions, such as *lhr71c* and *twotone*, the bars corresponding to MUMPs tend to grow taller as the number of CPUs increases. Both codes use similar pre-ordering algorithms, and although individually different, the fill-in and operation count are comparable on an average on a single CPU. As the number of CPU's increases, the operation count in WSMP tends to increase gradually due to added load-balancing constraints on the ordering. MUMPS' ordering, however, is identical on 1 to 4 CPUs, but shows a marked degradation on 8 (or more) CPUs.

The comparison of WSMP with Super_LU$_{Dist}$ in Figure 4 reveals, Super_LU$_{Dist}$ is slower by a factor ranging from 2 to 30 on a single CPU. However, it tends to achieve higher speedups and the bars in Figure 4 tend to grow shorter as the number of CPUs increases. This is not surprising because Super_LU$_{Dist}$ does not incur the communication overhead of dynamic pivoting and it is always easier to obtain good speedups by a slow serial program than by a fast one. Despite better speedups, with the exception of *onetone1*, WSMP was still significantly faster than Super_LU$_{Dist}$ on up to 8 CPUs.

Figure 5 shows the speedup of WSMP on two categories of matrices—those that are more than 90% symmetric and those that are less than 33% symmetric. The speedup patterns for the nearly symmetric matrices appear quite similar, with all achieving a speedup between 4 and 5 on 8 CPUs. However, it is more erratic for the highly unsymmetric matrices. In particular, very unstructured matrices arising in optimization problems achieve poor speedups because their task- and data-DAGs tend to have a large number of edges, with many of them connecting distant portions of the DAGs. This increases the overheads due to communication and load-imbalance. There does not appear to be any identifiably best ratio of threads to processes in an 8 CPU configuration. The small variations in run times appear to be the result of differences in fill-in and DAG structures due to different orderings produced by a parallel nested-dissection code whose output is sensitive to the processor configuration.

4 Concluding Remarks and Future Work

In this paper, we introduce WSMP's shared- and distributed-memory parallel sparse direct solver for general matrices and compare its performance with that of MUMPS and Super_LU$_{Dist}$ for 24 test matrices on 1, 2, 4, and 8 CPUs. Out of the 96 comparisons with MUMPS shown in Figures 2 and 3, there are only six cases where MUMPS is faster. Similarly, Super_LU$_{Dist}$ is faster than WSMP in only two of the 96 cases. On an average, WSMP appears to be significantly faster that both other distributed-memory parallel solvers, which themselves are the among best such solvers available. However, as discussed in Section 3, the parallel unsymmetric solver still needs improvement and tuning, particularly for very unsymmetric and unstructured matrices, such as those arising in linear programming problems.

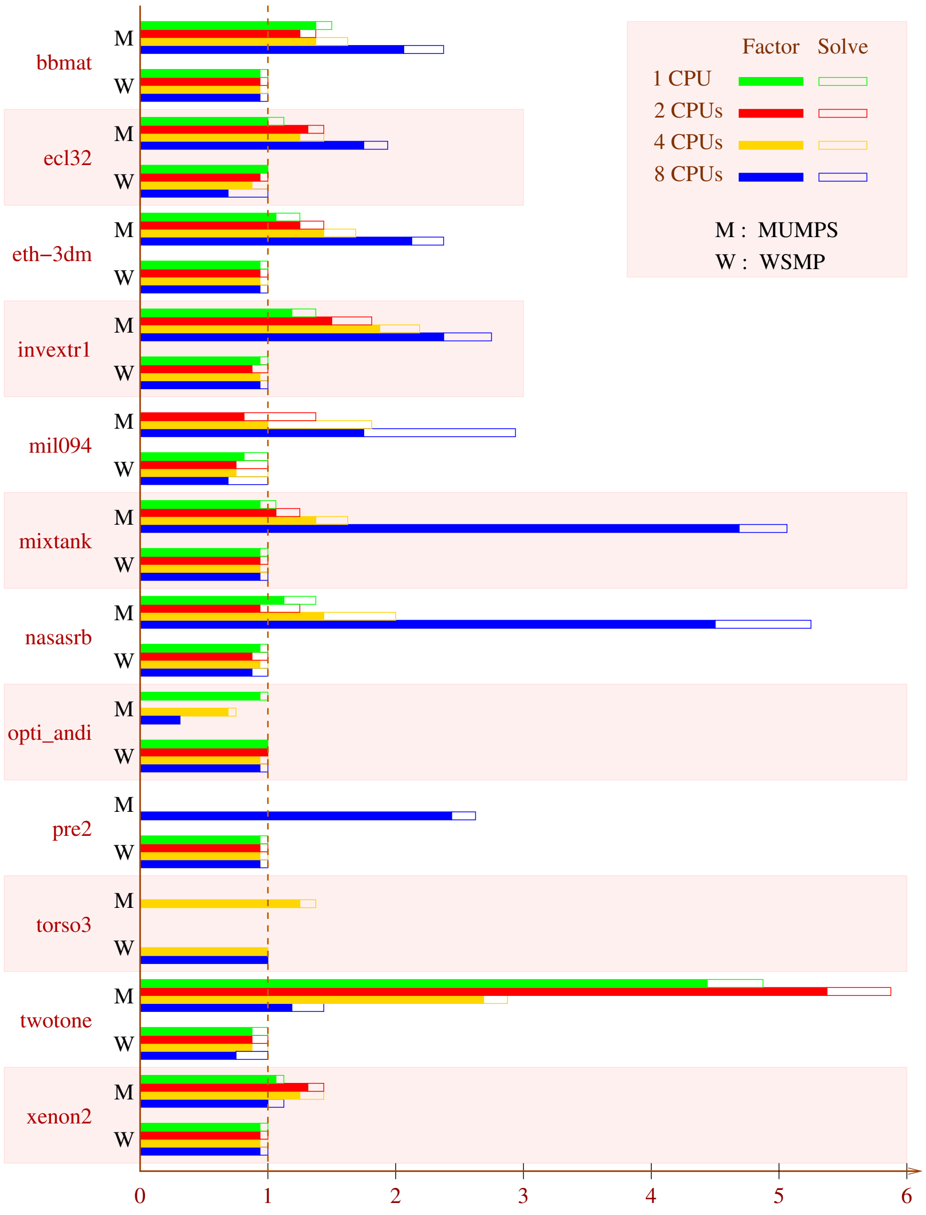


Fig. 2. A comparison of the factor and solve time of WSMP and MUMPS for the 12 largest systems of the test suite on 1, 2, 4, and 8 CPUs. All times are normalized with respect to the time taken by WSMP. Furthermore, the times spent by both packages in the factorization and solve (including iterative refinement) phases are denoted by filled and empty bars, respectively

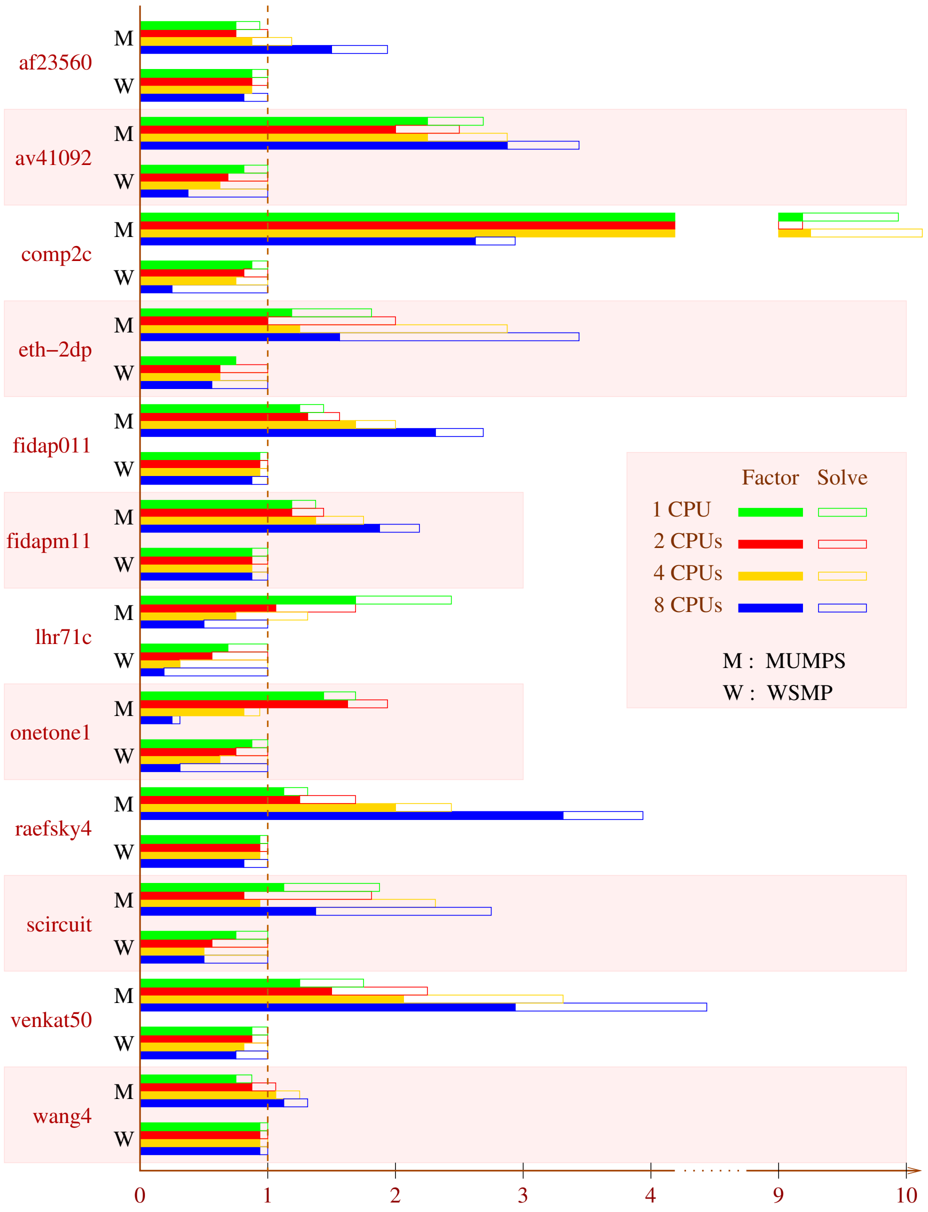


Fig. 3. A comparison of the factor and solve times of WSMP and MUMPS for the 12 smaller systems of the test suite on 1, 2, 4, and 8 CPUs. All times are normalized with respect to the time taken by WSMP. Furthermore, the times spent by both packages in the factorization and solve (including iterative refinement) phases are denoted by filled and empty bars, respectively

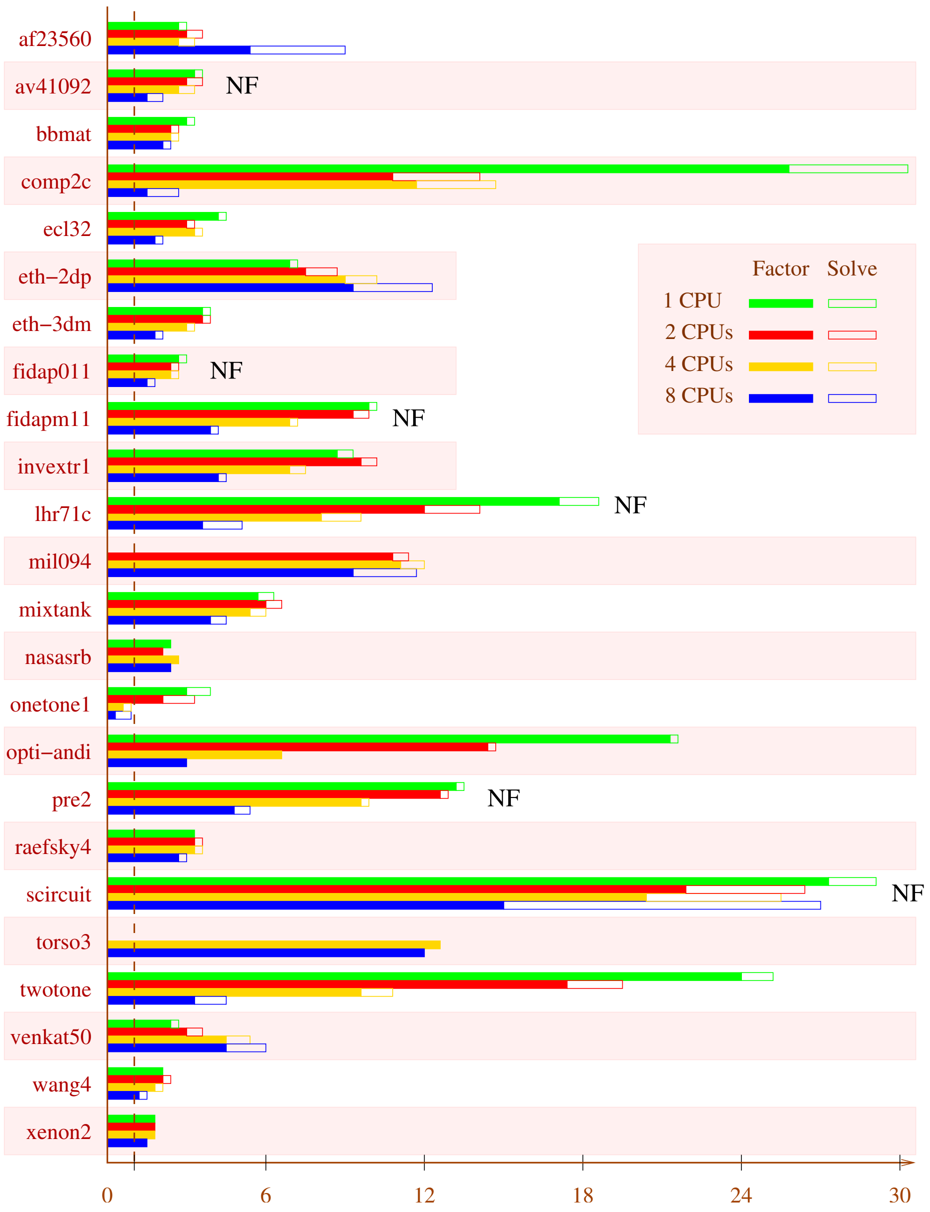


Fig. 4. A comparison of the time taken by Super_LU$_{Dist}$ to factor and solve 24 systems of the test suite on 1, 2, 4, and 8 CPUs with respect to the time taken by WSMP, which is assumed to be one unit in each case. The cases marked "NF" are those in which Super_LU$_{Dist}$ generated an incorrect solution due to absence of partial pivoting

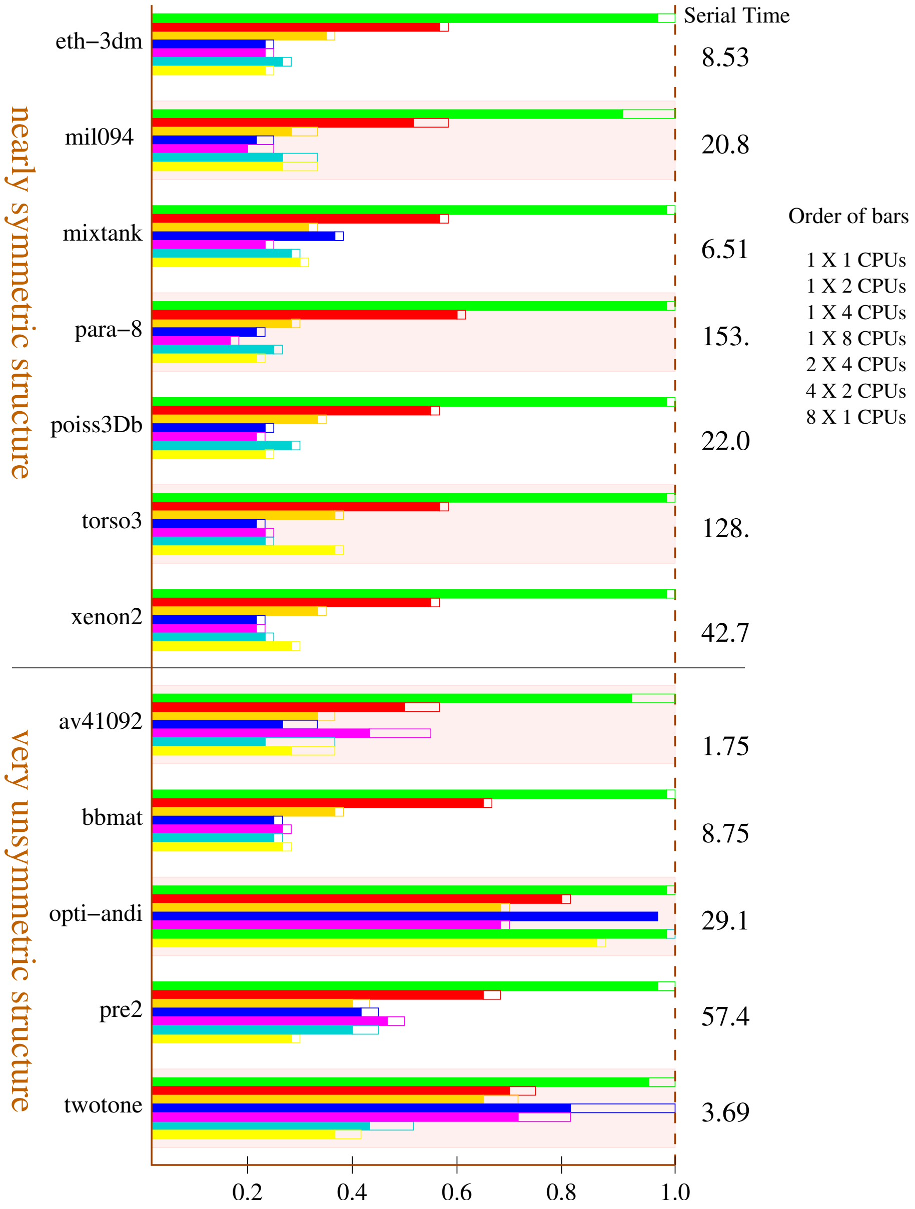


Fig. 5. WSMP factor and solve times from 1 to 8 CPUs (normalized w.r.t. single CPU time). The 8-CPU results are presented for four different configurations $t \times p$, where t is the number of threads and p is the number of MPI processes

References

1. Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

2. Patrick R. Amestoy, Iain S. Duff, Jacko Koster, and J. Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.
3. Timothy A. Davis and Iain S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM Journal on Matrix Analysis and Applications*, 18(1):140–158, 1997.
4. Iain S. Duff and Jacko Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22(4):973–996, 2001.
5. Anshul Gupta. A high-performance GEPP-based sparse solver. In *Proceedings of PARCO*, 2001. *http://www.cs.umn.edu/˜agupta/doc/parco-01.ps*.
6. Anshul Gupta. Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 24(2):529–552, 2002.
7. Anshul Gupta. WSMP: Watson sparse matrix package (Part-II: direct solution of general sparse systems). Technical Report RC 21888 (98472), IBM T. J. Watson Research Center, Yorktown Heights, NY, November 20, 2000. *http://www.cs.umn.edu/˜agupta/wsmp*.
8. Anshul Gupta. Recent advances in direct methods for solving unsymmetric sparse systems of linear equations. *ACM Transactions on Mathematical Software*, 28(3):301–324, September 2002.
9. Steven M. Hadfield. *On the LU Factorization of Sequences of Identically Structured Sparse Matrices within a Distributed Memory Environment*. PhD thesis, University of Florida, Gainsville, FL, 1994.
10. Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29(2):110–140, 2003.
11. Jurgen Schulze. Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods. *Bit Numerical Mathematics*, 41(4):800–841, 2001.

Simple and Efficient Modifications of Elimination Orderings

Pinar Heggernes and Yngve Villanger

Department of Informatics, University of Bergen, N-5020 Bergen, Norway
{pinar,yngvev}@ii.uib.no

Abstract. We study the problem of modifying a given elimination ordering through local reorderings. We present new theoretical results on equivalent orderings, including a new characterization of such orderings. Based on these results, we define the notion of k-optimality for an elimination ordering, and we describe how to use this in a practical context to modify a given elimination ordering to obtain less fill. We experiment with different values of k, and report on percentage of fill that is actually reduced from an already good initial ordering, like Minimum Degree.

1 Introduction

One of the most important and well studied problems related to sparse Cholesky factorization is to compute elimination orderings that give as few nonzero entries as possible in the resulting factors. In graph terminology, the problem can be equivalently modeled through the well known *Elimination Game* [10,13], where the input graph G corresponds to the nonzero structure of a given symmetric positive definite matrix A, and the output graph G^+ corresponds to the Cholesky factor of A. The set of edges that is added to G to obtain G^+ is called *fill*, and different output graphs with varying fill are produced dependent on the elimination ordering in which the vertices are processed. Computing an ordering that minimizes the number of fill edges is an NP-hard problem [15], and hence various heuristics are proposed and widely used, like Minimum Degree, Nested Dissection, and combinations of these.

Sometimes, after a good elimination ordering with respect to fill is computed by some heuristic, it is desirable to do some modifications on this ordering in order to achieve better properties for other performance measures, like storage or parallelism, without increasing the size of fill. Elimination orderings that result in the same filled graph are called *equivalent orderings*, and they can be used for this purpose [8]. In this paper, we prove some properties of equivalent orderings, and we give a new characterization of such orderings based on local permutations. These results are useful in practice when one seeks to modify a given ordering in order to group subsets of vertices close to or far from each other without changing fill.

Based on the mentioned results, we then move on to orderings that are not equivalent to the given ordering, and we study how local permutations of consecutive vertices in a given elimination ordering can affect the resulting fill. We define the notion of k-*optimality* for an elimination ordering, based on vertex subsets of size k. We give theoretical results on k-optimality, and we conclude with an algorithm that starts with any given ordering and makes it k-optimal for a chosen k. Finally, we implement a relaxed variant of this algorithm to exhibit its practical potential. For an efficient implementation, we use a data structure suitable for chordal graphs, called *clique trees*. Our tests show that even when we start with a good ordering, like Minimum Degree, substantial amount of fill can be reduced by trying to make the given ordering 3-optimal.

2 Preliminaries

A graph is denoted by $G = (V, E)$, where V is the set of vertices with $|V| = n$, and $E \subseteq \binom{V}{2}$ is the set of edges with $|E| = m$. When a graph G is given, we will use $V(G)$ and $E(G)$ to denote the vertices and edges of G respectively. The *neighbors* of a vertex v in G are denoted by the set $N_G(v) = \{u \mid uv \in E\}$, and the *closed neighborhood* of v is $N_G[v] = N_G(v) \cup \{v\}$. For a given vertex subset $A \subseteq V$, $G(A)$ denotes the graph induced by the vertices of A in G. A is a *clique* if $G(A)$ is a complete graph. A vertex v is called *simplicial* if $N_G(v)$ is a clique.

A *chord* of a cycle is an edge connecting two non-consecutive vertices of the cycle. A graph is *chordal* if it contains no induced chordless cycle of length ≥ 4. A graph $H = (V, E \cup F)$ is called a *triangulation* of $G = (V, E)$ if H is chordal.

An *elimination ordering* of G is a function $\alpha : V \leftrightarrow \{1, 2, ..., n\}$. We will say that $\alpha(v)$ is the *position* or *number* of v in α, and also use the equivalent notation $\alpha = \{v_1, v_2, ..., v_n\}$ meaning that $\alpha(v_i) = i$. For the opposite direction we use $\alpha^{-1}(i) = v_i$ to find a vertex, if the number in the ordering is given and $\alpha = \{v_1, v_2, ..., v_n\}$. The following algorithm simulates the production of fill in Gaussian elimination [10,13]:

```
**Algorithm** Elimination Game
**Input:** A graph $G = (V, E)$ and an ordering $\alpha = \{v_1, ..., v_n\}$ of $V$.
**Output:** The filled graph $G^+_\alpha$.
$G^0_\alpha = G$;
**for** $i = 1$ **to** $n$ **do**
    Let $F^i$ be the set of edges necessary to make $N_{G^{i-1}_\alpha}(v_i)$ a clique;
    Obtain $G^i_\alpha$ by adding the edges in $F^i$ to $G^{i-1}_\alpha$ and removing $v_i$;
$G^+_\alpha = (V, E + \cup_{i=1}^n F^i)$;
```

We will call G^i_α the i^{th} *elimination graph*. During Elimination Game, two vertices u and v that are not yet processed become *indistinguishable* after step i if $N_{G^i_\alpha}[u] = N_{G^i_\alpha}[v]$, and they remain indistinguishable until one of them is eliminated [7]. Given G and α, we will say that u and v are indistinguishable if u and v are indistinguishable at step $\min\{\alpha(u), \alpha(v)\}$, and otherwise we will say that they are *distinguishable*. The edges that are added to G during this algorithm are called *fill edges*, and uv is an edge in G^+_α if and only if $uv \in E$ or there exists a path $u, x_1, x_2, ..., x_k, v$ in G where $\alpha(x_i) < \min\{\alpha(u), \alpha(v)\}$, for $1 \leq i \leq k$ [12].

If no fill edges are added during Elimination Game, $G^+_\alpha = G$ and α is called a *perfect elimination ordering (peo)*. This corresponds to repeatedly removing a simplicial vertex until the graph becomes empty. A graph is chordal if and only if it has a peo [5]; thus filled graphs are exactly the class of chordal graphs, and Elimination Game is a way of producing triangulations of a given graph. Every chordal graph has at least two non-adjacent simplicial vertices [4], and hence any clique of size k in a chordal graph can be ordered consecutively with numbers $n-k+1, n-k+2, ..., n$ in a peo. An elimination ordering is *minimum* if no other ordering can produce fewer fill edges. Ordering α is a *minimal elimination ordering* if no proper subgraph of G^+_α is a triangulation of G, and G^+_α is then called a *minimal triangulation* of G. While it is NP-hard to compute minimum fill [15], minimal fill can be computed in polynomial time [12]. Minimal fill can in general be far from minimum, however algorithms exist to make a given elimination ordering minimal by removing edges from the filled graph [1,3,11]. For our purposes, when modifying a given elimination ordering to reduce fill, we will always start by making the given ordering minimal by removing the redundant edges in the filled graph using an appropriate existing algorithm. Two elimination orderings α and β are called *equivalent* if $G^+_\alpha = G^+_\beta$ [8]. If α is a minimal elimination ordering then every peo β of G^+_α is equivalent to α [1].

For an efficient implementation of the ideas presented in this paper, we need the notion of clique trees defined for chordal graphs [6]. A *clique tree* of a chordal graph G is a tree T, whose vertex set is the set of maximal cliques of G, that satisfies the following property: for every vertex v in G, the set of maximal cliques containing v induces a connected subtree of T. We will be working on a clique tree of G^+_α to identify groups of vertices that can be locally repermuted to give less fill. Clique trees can be computed in linear time [2]. A chordal graph has at most n maximal cliques [4], and thus a clique tree has $O(n)$ vertices and edges. We refer to the vertices of T as *tree nodes* to distinguish them from the vertices of G. Each tree node of T is thus a maximal clique of G.

3 Equivalent Orderings

Equivalent orderings of a given ordering were studied by Liu to achieve better properties for storage without increasing the amount of fill [8]. He used elimination tree rotations to compute appropriate equivalent orderings. Here, we will use repeated swapping of consecutive vertices.

Lemma 1. *Given G and α, let u and v be two vertices in G satisfying $\alpha(u) = \alpha(v) - 1 = i$. Let β be the ordering that is obtained from α by swapping u and v so that $\beta(u) = \alpha(v)$, $\beta(v) = \alpha(u)$, and $\beta(x) = \alpha(x)$ for all vertices $x \neq u, v$. Then $G^+_\alpha = G^+_\beta$ if and only if u and v are non-adjacent in G^+_α or indistinguishable in α.*

Proof. (If) If u and v are non-adjacent then it follows from Lemma 3.1 of [14] that $G^+_\alpha = G^+_\beta$. Let us consider the case of indistinguishable vertices. The graph G^+_α is obtained by picking the next vertex v_j in α and then making $N_{G^{j-1}_\alpha}(v_j)$ into a clique and removing v_j from G^{j-1}_α to obtain G^j_α, as described in Elimination Game. This can be considered as making n different sets of vertices into cliques. For every vertex $z \in V \setminus \{u, v\}$, its neighborhood which is made into a clique in G^+_α is the same as its neighborhood which is made into a clique in G^+_β. This follows from Lemma 4 of [12], since the same set of vertices are eliminated prior to z in both orderings. Now we have to show that eliminating u and then v results in the exact same set of fill edges as eliminating v and then u in the graph $G_\alpha^{i-1} = G_\beta^{i-1}$. Vertices u and v are indistinguishable, so $N_{G_\alpha^{i-1}}[u] = N_{G_\beta^{i-1}}[v]$ which is made into a clique in both cases. Observe that $N_{G_\alpha^i}[v] \subseteq N_{G_\alpha^{i-1}}[u]$ and that $N_{G_\beta^i}[u] \subseteq N_{G_\beta^{i-1}}[v]$, thus making these sets into cliques will not introduce any new edges. Thus $G_\alpha^+ = G_\beta^+$ if u and v are indistinguishable.

(Only if) We show that the same filled graph is not obtained by swapping u and v when u and v are adjacent and distinguishable. Since u and v are adjacent and distinguishable in α, then we know that $N_{G_\alpha^{i-1}}[u] \neq N_{G_\alpha^{i-1}}[v]$. Thus there exists a vertex $x \in N_{G_\alpha^{i-1}}[u] \setminus N_{G_\alpha^{i-1}}[v]$ or there exists a vertex $y \in N_{G_\alpha^{i-1}}[v] \setminus N_{G_\alpha^{i-1}}[u]$. Observe also that $N_{G_\alpha^{i-1}}[u] = N_{G_\beta^{i-1}}[u]$ and that $N_{G_\alpha^{i-1}}[v] = N_{G_\beta^{i-1}}[v]$, since the exact same set of vertices has been eliminated before u and v in both α and β, and thus $G_\alpha^{i-1} = G_\beta^{i-1}$. If vertex x exists, then it follows from Lemma 4 of [12] that edge $xv \in G_\alpha^+$ and $xv \notin G_\beta^+$. If vertex y exists, then it also follows from Lemma 4 of [12] that edge $yu \in G_\beta^+$ and $yu \notin G_\alpha^+$. Thus $G_\alpha^+ \neq G_\beta^+$ if u and v are adjacent and distinguishable.

Observe that if u and v are indistinguishable in α they are also indistinguishable in β. Note that similar results have been shown for the degrees of such consecutive vertices u and v [7]. Based on the above result, we give an interesting characterization of equivalent orderings in Theorem 1, which can be used as an algorithm to modify a given ordering and generate any desired equivalent ordering. For the proof of Theorem 1 we first need the following lemma.

Lemma 2. *Let $k < n$ be a fixed integer and let α and β be two equivalent orderings on G such that $\alpha^{-1}(i) = \beta^{-1}(i)$ for $1 \leq i \leq k$. Let v be the vertex that satisfies $\beta(v) = k+1$ and $\alpha(v) = k+d$, $d > 1$, and let u be the predecessor of v in α so that $\alpha(u) = \alpha(v) - 1$. Obtain a new ordering σ from α by only swapping u and v. Then $G_\sigma^+ = G_\beta^+$.*

Proof. Note first that since α and β coincide in the first k vertices, and v is the vertex numbered $k+1$ in β, the position of v must be larger than $k+1$ in α, as stated in the premise of the lemma. Note also for the rest of the proof that $\alpha(u) = k+d-1 > k$. We need to show that u and v are either non-adjacent in $G_\alpha^+ = G_\beta^+$ or indistinguishable in α, as the result then follows immediately from Lemma 1. Assume on the contrary that $uv \in E(G_\alpha^+) = E(G_\beta^+)$ and that u and v are distinguishable in α. Thus there exists at least a vertex $x \in N_{G_\alpha^{k+d-1}}(u) \setminus N_{G_\alpha^{k+d-1}}[v]$, or a vertex $y \in N_{G_\alpha^{k+d-1}}(v) \setminus N_{G_\alpha^{k+d-1}}[u]$. If vertex x exists, then $x \notin N_{G_\alpha^{k+d-1}}(v)$, and thus $x \notin N_{G_\alpha^k}(v) = N_{G_\beta^k}(v)$. It follows that $vx \notin E(G_\beta^+)$, since v is eliminated at step $k+1$ and before u and x in β. This contradicts our assumption that $G_\alpha^+ = G_\beta^+$, because $vx \in G_\alpha^+$ since $ux, uv \in E(G_\alpha^+)$, and u is eliminated before v and x. If vertex y exists, then $uy \notin E(G_\alpha^+)$, since $y \notin N_{G_\alpha^{k+d-1}}(u)$, and u is eliminated before v and y in α. Vertex v is eliminated before y and u in β, and this gives the same contradiction as above, since uv and vy are contained in $E(G_\beta^+)$, and thus $uy \in E(G_\beta^+)$.

Regarding orderings β and σ in Lemma 2, note in particular that $\sigma^{-1}(i) = \beta^{-1}(i)$ for $1 \leq i \leq k$ and $\sigma(v) = k+d-1$. Thus β and σ coincide in the first k positions and

v has moved one position closer to its position in β. We are now ready to present our new characterization of equivalent orderings.

Theorem 1. *Two orderings α and β on G are equivalent, i.e., $G_\alpha^+ = G_\beta^+$, if and only if β can be obtained by starting from α and repeatedly swapping pairs of consecutive non-adjacent or indistinguishable vertices.*

Proof. The if part of the theorem follows directly from Lemma 1. We prove the only if part by showing that it is always possible to make two equivalent orderings identical by repeatedly swapping pairs of consecutive non-adjacent or indistinguishable vertices in one of them. For this we use the following inductive approach.

Induction hypothesis: Let α and β be two equivalent orderings on G. Modify α so that the k first vertices of α coincide with the k first vertices of β, for a $k \in [0, n]$, and the remaining $n - k$ vertices have the same local relative order as before the change. Then $G_\alpha^+ = G_\beta^+$.

Base case: When $k = 0$ α is unchanged and thus $G_\alpha^+ = G_\beta^+$.

Induction step: Given two equivalent orderings α and β on G such that the k first vertices in α coincide with the k first vertices in β. We have to show that, in α, one of the vertices in positions $k + 1, k + 2, ..., n$ can be moved to position $k + 1$ so that the vertex in position $k+1$ is the same in α and in β. We will show that this can be achieved by repeatedly swapping pairs of consecutive non-adjacent or indistinguishable vertices in positions $k + 1, k + 2, ..., n$ of α. We use induction also for this purpose.

Induction hypothesis: Given two equivalent orderings α and β on G such that the k first vertices in α coincide with the k first vertices in β. Let v be vertex number $k + 1$ in β, and let $\alpha(v) = k + d$, for a $d \in [1, n - k]$. Move v so that $\alpha(v) = k + d - i$, for an $i \in [0, d - 1]$. Then $G_\alpha^+ = G_\beta^+$.

Base case: Let $i = 0$, then α is unchanged and thus $G_\alpha^+ = G_\beta^+$.

Induction step: Given two equivalent orderings α and β on G such that the k first vertices in α coincide with the k first vertices in β, and vertex v which is numbered $k+1$ in β has number $k + d - i$ in α. It follows from Lemma 2 that v can be swapped with its predecessor in α and thus moved to position $k + d - (i + 1)$ in α and G_α^+ remains unchanged.

We have thus proved that, the first vertex v of β which does not coincide with the vertex in the same position of α, can be repeatedly swapped with its predecessor in α until it reaches the same position in α as in β without changing the filled graph, and that this can again be repeated for all vertices. By Lemma 1 all swapped vertices are non-adjacent and indistinguishable, which completes the proof.

4 *k*-Optimal Elimination Orderings

In this section, we put the results of the previous section in a practical context with the purpose of modifying a given ordering α to reduce the resulting fill. As we have seen, swapping two consecutive non-adjacent or indistinguishable vertices of α does not change the fill. Consequently, if we want to swap consecutive vertices to reduce fill, these should be distinguishable and neighbors in G_α^+. The reason why we only consider vertices that appear consecutively in an elimination ordering is that it is then easy to do a local computation to find the change in the number of fill edges, without having to recompute the whole filled graph. This saves time for practical implementations. The following lemma gives the change in the number of fill edges when two adjacent vertices are swapped.

Lemma 3. *Given G and α, let uv be an edge of G^+_α such that $\alpha(u) = \alpha(v) - 1 = i$. Let β be the ordering that is obtained from α by swapping u and v. Then $|E(G^+_\alpha)| - |E(G^+_\beta)| = |N_{G^{i-1}_\alpha}(u)| - |N_{G^{i-1}_\alpha}(v)|$.*

Proof. Consider the elimination graph G^{i-1}_α. Since uv is an edge of G^+_α, it is also an edge of G^{i-1}_α. If we at step i eliminate u (which corresponds to ordering α), the set of fill edges added incident to v by u is $N_{G^{i-1}_\alpha}[u] \setminus N_{G^{i-1}_\alpha}[v]$. If on the other hand, v is eliminated at step i (which corresponds to ordering β), the set of fill edges added adjacent to u by v is $N_{G^{i-1}_\alpha}[v] \setminus N_{G^{i-1}_\alpha}[u]$. It follows from Lemma 4 of [12] that no other fill is changed since the set of vertices eliminated previous to every vertex $z \in V \setminus \{u, v\}$ are the same for α and β. Thus the difference is $|N_{G^{i-1}_\alpha}[u]| - |N_{G^{i-1}_\alpha}[v]| = |N_{G^{i-1}_\alpha}(u)| - |N_{G^{i-1}_\alpha}(v)|$.

Although uv is an edge of G^+_α, it might be that u and v do not appear consecutively in α but they appear consecutively in an equivalent ordering β. Since both α and β result in the same filled graph G^+_α, we want to be able to check all edges uv that appear consecutively in α or in an equivalent ordering β. Indeed, we will generalize this idea from 2 to k. Instead of only considering pairs of adjacent vertices, we will consider cliques K of size k whose vertices appear consecutively in the given ordering α or in an equivalent ordering β, and examine whether a local permutation of the vertices of K can reduce the resulting fill. Note that the vertices of any k-clique in a chordal graph can be ordered with numbers $n - k + 1, n - k + 2, ..., n$ in a peo [13], and thus we can always find an equivalent ordering β in which the vertices of a given k-clique are ordered consecutively. As a consequence of the results of the previous section, β can be computed from α by swapping pairs of consecutive non-adjacent or distinguishable vertices. However, since K is a clique in G^+_α, we will show that we are able to compute β more efficiently using a clique tree of G^+_α, rather than this more general characterization. Before that, we summarize the above ideas with a definition and an algorithm.

We will always start with making the given ordering α minimal by removing unnecessary edges from G^+_α by using an appropriate algorithm for this purpose. When α is minimal, we know that no ordering can result in a filled graph that is a proper subgraph of G^+_α. Thus reducing the number of fill edges further can only be achieved if some fill edges are removed and some new fill edges are introduced.

Definition 1. *An elimination ordering α is k-*optimal *on G if obtaining an equivalent ordering β where the vertices of K appear consecutively, and then locally permuting the vertices of K, cannot result in less fill for any k-clique K in G^+_α.*

```
**Algorithm** Minimal $k$-optimal ordering
**Input:** A graph $G$ and an ordering $\alpha$. ($\beta = \alpha$;)
**Output:** A minimal elimination ordering $\beta$ of $G$ where $|E(G^+_\beta)| \leq |E(G^+_\alpha)|$.
**repeat**
    Compute a minimal ordering $\sigma$ such that $E(G^+_\sigma) \subseteq E(G^+_\beta)$;
    $\beta =$ Compute-$k$-optimal$(G, \sigma)$;
**until** $\beta$ is minimal and $k$-optimal
```

The call Compute-k-optimal(G, σ) returns a k-optimal ordering based on σ. Note that computing a minimal ordering removes the unnecessary fill edges from the filled graph, and never adds new fill edges. Computing a k-optimal ordering, however, will both remove and add fill edges, and thus the resulting ordering after this step is not necessarily minimal. By the definition of k-optimality, this algorithm always produces an ordering β with less fill than that of α unless α is already minimal and k-optimal.

Simple examples exist which show that a k-optimal ordering is not necessarily $(k-1)$-optimal or $(k+1)$-optimal. However, we suspect that when checking k-optimality, if we also check all maximal cliques in G_α^+ of size less than k, then we can also guarantee $(k-1)$-optimality. Note that the number of maximal cliques in G_α^+ is $O(n)$, and thus this extra work is substantially less than the work of checking all (both maximal and non-maximal) k-cliques that the definition requires. We leave this as an open question.

We now describe a practical implementation of a variant of the above algorithm to exhibit the potential of the proposed ideas. Given G and σ, in order to implement the function Compute-k-optimal(G, σ) correctly, we need to be able to do the following subtasks for each k-clique K of G_σ^+: 1. Compute every equivalent ordering where the vertices of K are ordered consecutively. 2. Try all possible local permutations of the vertices of K in each of these equivalent orderings. Recall however that we need not consider k-cliques in which all vertices are pairwise indistinguishable, and we need not consider local permutations that result in equivalent orderings. Still, the remaining work is too much for a practical algorithm, and in the practical version we generate several but *not* all equivalent orderings in which the vertices of K are ordered consecutively. Furthermore, we only test a single local permutation of K; one that we suspect might give the highest fill reduction. For both of these subtasks, we use clique trees. The vertices of every k-clique K must appear together in at least one tree node of any clique tree of G_σ^+, and subsets of K that appear in other tree nodes will give us various local permutations that we will try. Then using the clique tree, it is easy to compute orderings where the vertices of K appear together and follow the chosen permutations.

Observe that every leaf in any clique tree of G_α^+ contains a simplicial vertex, which can be eliminated as the first vertex in an equivalent ordering. Let us consider how we construct equivalent orderings for a clique K. Let T be a clique tree of G_α^+. A subtree or a forest of subtrees remains when we remove every tree node in T containing K. One equivalent ordering is now created for each remaining subtree. Let T' be one such subtree, and let T'' be the rest of T when this subtree is removed. Let V' be the set of vertices of G contained in the tree nodes of T' and not contained in the tree nodes of T''. An equivalent ordering is now obtained by eliminating the vertices of $V \setminus (V' \cup K)$, then the vertices of K, and finally the vertices of V'. Within each of these three sets we eliminate the vertices in a simplicial order.

There exists at least one vertex in K which is not adjacent to any vertex in V'. If this was not the case, then K would be contained in the tree node of T' which is adjacent in T to a tree node containing K, which is a contradiction since none of the tree nodes of T' contains K. Eliminating such a vertex will make the remaining vertices of K into a clique. We want to avoid this for two reasons. The first is that we want to obtain a new graph, and the second is that we do not want to destroy the possibility of removing edges in K. A simple solution to this problem is to eliminate these vertices last when K is

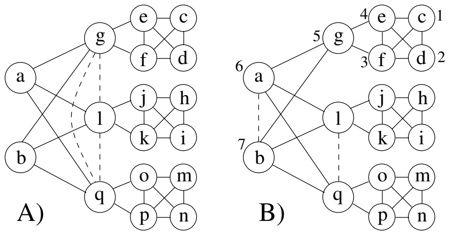


Fig. 1. The solid edges are the original edges of the given graph, while the dashed edges are fill edges resulting from the elimination process. In figure A we use the elimination ordering given by the alphabetic ordering of the vertices, which is minimal and also a Minimum Degree ordering. Consider the equivalent ordering $\{c, d, e, f, g, a, b, h, i, ..., q\}$ and the clique $\{a, g\}$. Swapping a and g gives us a new ordering $\{c, d, e, f, a, g, b, h, i, ..., q\}$ which contains one less fill edge, as shown in figure B. Thus the Minimum Degree ordering of A is minimal but not 2-optimal

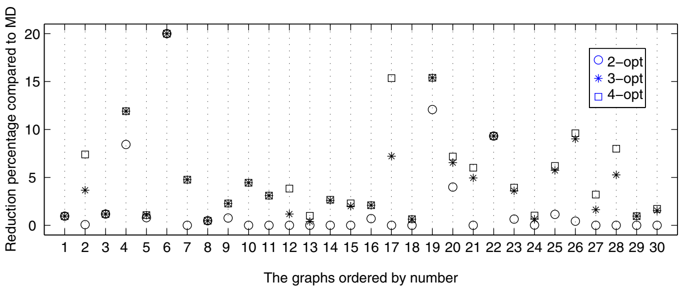


Fig. 2. This figure only shows the reduction in percentage

eliminated. So the new order is created as follows: Eliminate the vertices of $V \setminus (V' \cup K)$ in a simplicial order, then eliminate the vertices of K that are adjacent to a vertex in V' in a random order, then eliminate the rest of the vertices of K in any order. Finally eliminate the vertices of V' in a simplicial order. This will give us an ordering that introduces new fill edges, and possibly remove some.

In order to achieve good orderings, it is important to start the process with an ordering that already produces low fill. We have tested our implementation with Minimum Degree as the starting ordering. Minimum Degree orderings are in general neither minimal nor k-optimal for any $k > 1$, although they tend to be often minimal in practice [1,11]. However, even Minimum Degree orderings that are minimal need not be 2-optimal, as shown in Figure 1. Thus Minimum Degree is a good starting ordering for our purposes. Table 1 and Figure 2 show the reductions in fill achieved by our implementation when trying to make an initial Minimum Degree ordering k-optimal for $k \in \{2, 3, 4\}$ on a collection of sparse matrices downloaded from Matrix Market [9]. Columns 6, 8, and 10 of the table show the achieved reduction in the number of fill edges, whereas columns 7, 9, and 11 show the reduction in percentage. Figure 2 shows the reductions in percentage only. As can be seen, the reduction is substantial in several cases. For 4 other graphs

Table 1. We would like to remark that in order to avoid "lucky" reductions, we ran Minimum Degree on 5 random permutations of each graph, and chose the ordering that gave the lowest fill as the starting ordering

No	Name	# vertices	# edges	MD fill	2-opt	2-opt %	3-opt	3-opt %	4-opt	4-opt %
1	494_bus	494	586	314	3	0.96	3	0.96	3	0.96
2	ash292	292	958	1392	1	0.07	51	3.66	103	7.40
3	bcspwr03	118	179	85	1	1.18	1	1.18	1	1.18
4	bcspwr04	274	669	403	34	8.44	48	11.91	48	11.91
5	bcspwr05	443	590	375	3	0.80	4	1.07	4	1.07
6	bcsstk03	112	265	10	2	20.00	2	20.00	2	20.00
7	bcsstk05	153	1135	1197	0	0	57	4.76	57	4.76
8	bcsstk20	485	1328	412	2	0.49	2	0.49	2	0.49
9	bcsstk22	138	282	263	2	0.76	6	2.28	6	2.28
10	can__144	144	576	225	0	0	10	4.44	10	4.44
11	can__161	161	608	1551	0	0	48	3.09	48	3.09
12	can__187	187	652	1277	0	0	15	1.17	49	3.84
13	can__229	229	774	2008	0	0	8	0.40	20	1.00
14	can__256	256	1330	1913	0	0	50	2.61	51	2.67
15	can__268	268	1407	2096	0	0	42	2.00	48	2.29
16	can__292	292	1124	1143	8	0.70	24	2.10	24	2.10
17	dwt__162	162	510	638	0	0	46	7.21	98	15.36
18	dwt__193	193	1650	2367	0	0	15	0.63	15	0.63
19	dwt__198	198	602	513	62	12.09	79	15.40	79	15.40
20	dwt__209	209	767	1102	44	4.00	72	6.53	79	7.17
21	dwt__221	221	704	849	0	0	42	4.95	51	6.01
22	dwt__234	234	306	204	19	9.31	19	9.31	19	9.31
23	dwt__245	245	608	610	4	0.66	22	3.61	24	3.93
24	dwt__307	307	1108	3421	1	0.03	22	0.64	35	1.02
25	dwt__310	310	1069	1920	22	1.15	110	5.73	119	6.20
26	dwt__346	346	1443	3516	15	0.43	317	9.02	338	9.61
27	dwt__361	361	1296	3366	0	0	55	1.63	108	3.21
28	dwt__419	419	1572	3342	0	0	176	5.27	267	8.00
29	lshp_265	265	744	2185	0	0	21	0.96	21	0.96
30	lshp_406	406	1155	4074	0	0	63	1.55	70	1.72

that we tested, bcsstk04, lund_a, lund_b, and nos1, no reduction was achieved for these k-values, and to save space, we omit these graphs in the table and the following figure.

We would like to stress that the goal of our numerical experiments has been to exhibit the potential of the proposed algorithm to find good quality orderings with respect to fill.

We have not optimized the the running time of our code, thus we find it useless to give the running time of computing a 3-optimal or a 4-optimal ordering compared to the Minimum Degree algorithm whose code has been thoroughly optimized through the last decades. Although computing 3-optimal and 4-optimal orderings are computationally heavy tasks that require more time than Minimum Degree, we believe that practical implementations using approximations are possible. This is an interesting topic for further research.

References

1. J. R. S. Blair, P. Heggernes, and J. A. Telle. A practical algorithm for making filled graphs minimal. *Theor. Comput. Sci.*, 250:125–141, 2001.
2. J. R. S. Blair and B. W. Peyton. An introduction to chordal graphs and clique trees. In *Sparse Matrix Computations: Graph Theory Issues and Algorithms*, pages 1–30. Springer Verlag, 1993.
3. E. Dahlhaus. Minimal elimination ordering inside a given chordal graph. In Proceedings of WG 1997, pages 132–143. Springer Verlag, 1997. LNCS 1335.
4. G.A. Dirac. On rigid circuit graphs. *Anh. Math. Sem. Univ. Hamburg*, 25:71–76, 1961.
5. D.R. Fulkerson and O.A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Math.*, 15:835–855, 1965.
6. F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combin. Theory Ser. B*, 16:47–56, 1974.
7. J. A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981.
8. J. W. H. Liu. Equivalent sparse matrix reorderings by elimination tree rotations. *SIAM J. Sci. Stat. Comput.*, 9(3):424–444, 1988.
9. R. Boisvert, R. Pozo, K. Remington, B. Miller, and R. Lipman. *NIST Matrix Market*. http://math.nist.gov/MatrixMarket/.
10. S. Parter. The use of linear graphs in Gauss elimination. *SIAM Review*, 3:119–130, 1961.
11. B. W. Peyton. Minimal orderings revisited. *SIAM J. Matrix Anal. Appl.*, 23(1):271–294, 2001.
12. D. Rose, R.E. Tarjan, and G. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM J. Comput.*, 5:146–160, 1976.
13. D. J. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, 32:597–609, 1970.
14. Y. Villanger. Lex M versus MCS-M. Reports in Informatics 261, University of Bergen, Norway, 2004
15. M. Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM J. Alg. Disc. Meth.*, 2:77–79, 1981.

Optimization of a Statically Partitioned Hypermatrix Sparse Cholesky Factorization[⋆]

José R. Herrero and Juan J. Navarro

Computer Architecture Department, Universitat Politècnica de Catalunya
Jordi Girona 1-3, Mòdul D6, E-08034 Barcelona, Spain
{josepr,juanjo}@ac.upc.edu

Abstract. The sparse Cholesky factorization of some large matrices can require a two dimensional partitioning of the matrix. The sparse hypermatrix storage scheme produces a recursive 2D partitioning of a sparse matrix. The subblocks are stored as dense matrices so BLAS3 routines can be used. However, since we are dealing with sparse matrices some zeros may be stored in those dense blocks. The overhead introduced by the operations on zeros can become large and considerably degrade performance. In this paper we present an improvement to our sequential in-core implementation of a sparse Cholesky factorization based on a hypermatrix storage structure. We compare its performance with several codes and analyze the results.

1 Introduction

Sparse Cholesky factorization is heavily used in several application domains, including finite-element and linear programming algorithms. It forms a substantial proportion of the overall computation time incurred by those methods. Consequently, there has been great interest in improving its performance [8,20,22]. Methods have moved from column-oriented approaches into panel or block-oriented approaches. The former use level 1 BLAS while the latter have level 3 BLAS as computational kernels [22]. Operations are thus performed on blocks (submatrices). In this paper we address the optimization of the sparse Cholesky factorization of large matrices. For this purpose, we use a *Hypermatrix* [10] block data structure with static block sizes.

1.1 Background

Block sizes can be chosen either statically (fixed) or dynamically. In the former case, the matrix partition does not take into account the structure of the sparse matrix. In the latter case, information from the *elimination tree* [17] is used. Columns having similar structure are taken as a group. These column groups are called *supernodes* [18]. Some supernodes may be too large to fit in cache and it is advisable to split them into *panels* [20,22]. In other cases, supernodes can be too small to yield good performance. This is the case of supernodes with just a few columns. Level 1 BLAS routines are used in this case and the performance obtained is therefore poor. This problem can be reduced by *amalgamating*

[⋆] This work was supported by the Ministerio de Ciencia y Tecnología of Spain and the EU FEDER funds (TIC2001-0995-C02-01).

several supernodes into a single larger one [1]. Although, some null elements are then both stored and used for computation, the resulting use of level 3 BLAS routines often leads to some performance improvement.

1.2 Hypermatrix Representation of a Sparse Matrix

Sparse matrices are mostly composed of zeros but often have small dense blocks which have traditionally been exploited in order to improve performance [8]. Our approach uses a data structure based on a hypermatrix (HM) scheme [10,21]. The matrix is partitioned recursively into blocks of different sizes. The HM structure consists of *N* levels of submatrices. The top *N-1* levels hold pointer matrices which point to the next lower level submatrices. Only the last (bottom) level holds data matrices. Data matrices are stored as dense matrices and operated on as such. Null pointers in pointer matrices indicate that the corresponding submatrix does not have any non-zero elements and is therefore unnecessary. Figure 1 shows a sparse matrix and a simple example of corresponding hypermatrix with 2 levels of pointers.

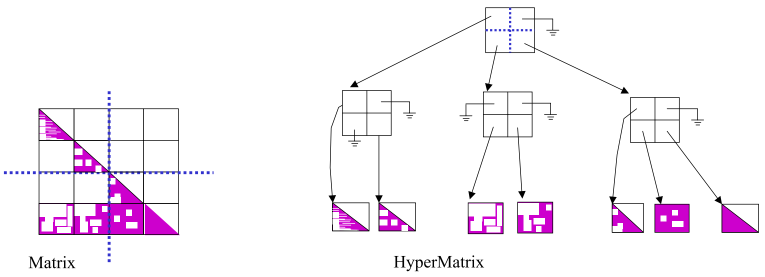


Fig. 1. A sparse matrix and a corresponding hypermatrix

The main potential advantages of a HM structure over 1D data structures, such as the Compact Row Wise structure, are: the ease of use of multilevel blocks to adapt the computation to the underlying memory hierarchy; the operation on dense matrices; and greater opportunities for exploiting parallelism. A commercial package known as PERMAS uses the hypermatrix structure [3]. It can solve very large systems out-of-core and can work in parallel. However, the disadvantages of the hypermatrix structure, namely the storage of and computation on zeros, introduce a large overhead. This problem can arise either when a fixed partitioning is used or when supernodes are amalgamated. In [2] the authors reported that a variable size blocking was introduced to save storage and to speed the parallel execution. In this way the HM was adapted to the sparse matrix being factored. The results presented in this paper, however, correspond to a static partitioning of the matrix into blocks of fixed sizes.

1.3 Machine and Matrix Characteristics

Execution took place on a 250 MHz MIPS R10000 Processor. The first level instruction and data caches have size 32 Kbytes. There is a secondary unified instruction/data cache with size 4 Mbytes. This processor's theoretical peak performance is 500 Mflops.

Table 1. Matrix characteristics

Matrix	Dimension	NZs	NZs in L	Density	Flops to factor
GRIDGEN1	330430	3162757	130586943	0.002	278891960665
QAP8	912	14864	193228	0.463	63764878
QAP12	3192	77784	2091706	0.410	2228094940
QAP15	6330	192405	8755465	0.436	20454297601
RMFGEN1	28077	151557	6469394	0.016	6323333044
TRIPART1	4238	80846	1147857	0.127	511884159
TRIPART2	19781	400229	5917820	0.030	2926231696
TRIPART3	38881	973881	17806642	0.023	14058214618
TRIPART4	56869	2407504	76805463	0.047	187168204525
pds1	1561	12165	37339	0.030	1850179
pds10	18612	148038	3384640	0.019	2519913926
pds20	38726	319041	10739539	0.014	13128744819
pds30	57193	463732	18216426	0.011	26262856180
pds40	76771	629851	27672127	0.009	43807548523
pds50	95936	791087	36321636	0.007	61180807800
pds60	115312	956906	46377926	0.006	81447389930
pds70	133326	1100254	54795729	0.006	100023696013
pds80	149558	1216223	64148298	0.005	125002360050
pds90	164944	1320298	70140993	0.005	138765323993

We have used several test matrices. All of them are sparse matrices corresponding to linear programming problems. QAP matrices come from Netlib [19] while others come from a variety of linear multicommodity network flow generators: A Patient Distribution System (PDS) [5], with instances taken from [9]; RMFGEN [4]; GRIDGEN [16]; TRIPARTITE [11]. Table 1 shows the characteristics of several matrices obtained from such linear programming problems. Matrices were ordered with METIS [14] and renumbered by an elimination tree postorder.

2 Performance Issues

In this section we present two aspects of the work we have done to improve the performance of our sparse Hypermatrix Cholesky factorization. Both of them are based on the fact that a matrix is divided into submatrices and operations are thus performed on blocks (submatrices).

A matrix M is divided into submatrices of arbitrary size. We call M_{br_i,bc_j} the data submatrix in block-row br_i and block-column bc_j. Figure 2 shows 3 submatrices within a matrix. The highest cost within the Cholesky factorization process comes from the multiplication of data submatrices. In order to ease the explanation we will refer to the three matrices involved in a product as A, B and C. For block-rows br_1 and br_2 (with $br_1 < br_2$), and block-column bc_j each of these blocks is $A \equiv M_{br_2,bc_j}$, $B \equiv M_{br_1,bc_j}$ and $C \equiv M_{br_2,br_1}$. Thus, the operation performed is $C = C - A \times B^t$, which means that submatrices A and B are used to produce an update on submatrix C.

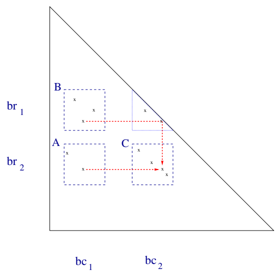


Fig. 2. Blocks within a matrix

2.1 Efficient Operation on Small Matrices

Choosing a block size for data submatrices is rather difficult. When operating on dense matrices, it is better to use large block sizes. On the other hand, the larger the block is, the more likely it is to contain zeros. Since computation with zeros is non productive, performance can be degraded. Thus, a trade-off between performance on dense matrices and operations on non-zeros must be reached. In a previous paper [12], we explained how we could reduce the block size while we improved performance. This was achieved by the use of a specialized set of routines which operate on small matrices of fixed size. By small matrices we mean matrices which fit in the first level cache. The basic idea used in producing this set of routines, which we call the Small Matrix Library (SML), is that of having dimensions and loop limits fixed at compilation time. For example, our matrix multiplication routines *mxmts_fix* outperform the vendor's BLAS (version 7.3) dgemm routine (*dgemm_nts*) for small matrices (fig. 3a) on an R10000 processor. We achieved similar results to outperform the ATLAS [23] dgemm routine.

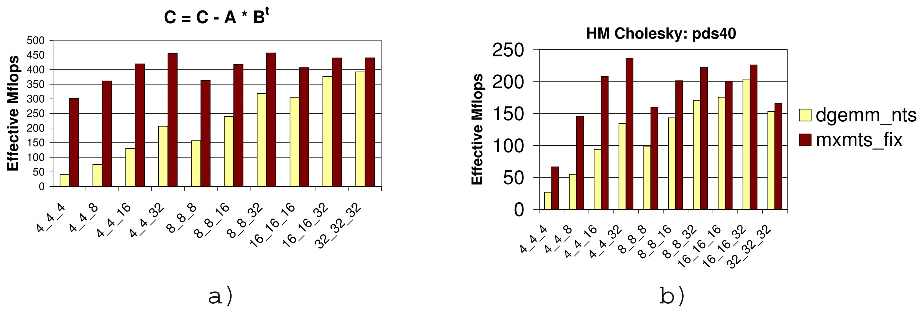


Fig. 3. a) Performance of different MxM^t routines for several matrix sizes. **b)** Factorization of matrix pds40: Mflops obtained by different MxM^t codes within HM Cholesky. Effective Mflops are reckoned excluding any operations on zeros

The matrix multiplication routine used affects the performance of hypermatrix Cholesky factorization. This operation takes up most of the factorization time. We found that when using *mxmts_fix*, a block size of 4×32 usually produced the best performance.

In order to illustrate this, figure 3b shows results of the HM Cholesky factorization on an R10000 for matrix pds40 [5]. The use of a fixed dimension matrix multiplication routine speeded up our Cholesky factorization an average of 12% for our test matrix set (table 1).

2.2 Reducing Overhead: Windows Within Data Submatrices

Let A and B be two off-diagonal submatrices in the same block-column of a matrix. At first glance, these matrices should always be multiplied since they belong to the same block-column. However, there are cases where it is not necessary. We are storing data submatrices as dense while the actual contents of the submatrix are not necessarily dense. Thus, the result of the product $A \times B^t$ can be zero. Such an operation will produce an update into some matrix C whenever there exists at least one column for which both matrices A and B have any non-zero element. Otherwise, if there are no such columns, the result will be zero. Consequently, that multiplication can be bypassed.

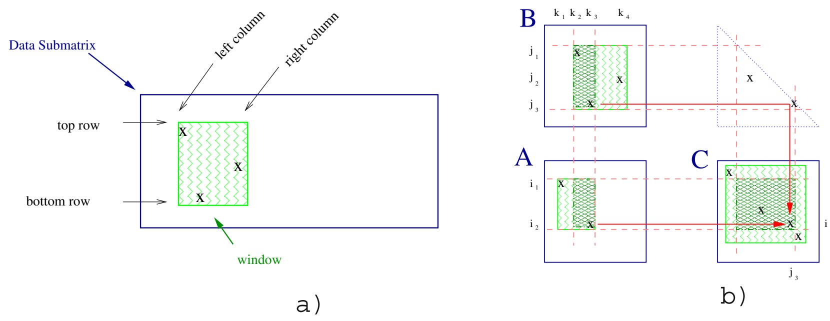


Fig. 4. a) A rectangular data submatrix and a window within it. **b)** Windows can reduce the number of operations

In order to reduce the storage and computation of zero values, we define *windows* of non-zeros within data submatrices. Figure 4a shows a window of non-zero elements within a larger block. The window of non-zero elements is defined by its top-left and bottom right corners. All zeros outside those limits are not used in the computations. Null elements within the window are still stored and computed. Storage of columns to the left of the window's leftmost column is avoided since all their elements are null. Similarly, we do not store columns to the right of the window's rightmost column. However, we do store zeros over the window's upper row and/or underneath its bottom row whenever these window's boundaries are different from the data submatrix boundaries, i.e. whole data submatrix columns are stored from the leftmost to the rightmost columns in a window. We do this to have the same leading dimension for all data submatrices used in the hypermatrix. Thus, we can use our specialized SML routines which work on matrices with fixed leading dimensions. Actually, we extended our SML library with routines which have the leading dimensions of matrices fixed, while the loop limits may be given as parameters. Some of the routines have all loop limits fixed, while others have only one, or two of them fixed. Other routines have all the loop limits given as parameters. The appropriate routine is chosen at execution time depending on the windows involved in the operation. Thus, although zeros can be stored above or underneath a window, they are not used for computation. Zeros can still exist within the window but, in general, the overhead is greatly reduced.

The use of windows of non-zero elements within blocks allows for a larger default hypermatrix block size. When blocks are quite full operations performed on them can be rather efficient. However, in those cases where only a few non-zero elements are present in a block, or the intersection of windows results in a small area, it is necessary to compute only a subset of the total block (dark areas within figure 4b). When the column-wise intersection of windows in matrices A and B is null, we can avoid the multiplication of these two matrices.

3 Results

Figure 5 shows the results obtained with several variants of our HM Cholesky code. In all cases the code follows a right looking scheduling with a fixed partitioning of the hypermatrix (the elimination tree is consequently not used at all). We use a 4×32 data matrix size as stated in section 2.1. The first and second bars allow for the evaluation of the usage of windows within data submatrices (in a 2D layout of such submatrices). The usage of windows clearly improves the performance of our sparse hypermatrix Cholesky algorithm. The second and third bars show the results of 2D and 1D data layouts and scheduling of the data submatrices (windows are used in both cases). The former uses upper levels in the HM with sizes 32×32 and 512×512. For the latter, we define only an upper level with sizes 32×32. A 2D data layout and scheduling of the computations is beneficial to the efficient computation of the Cholesky factorization of large sparse matrices.

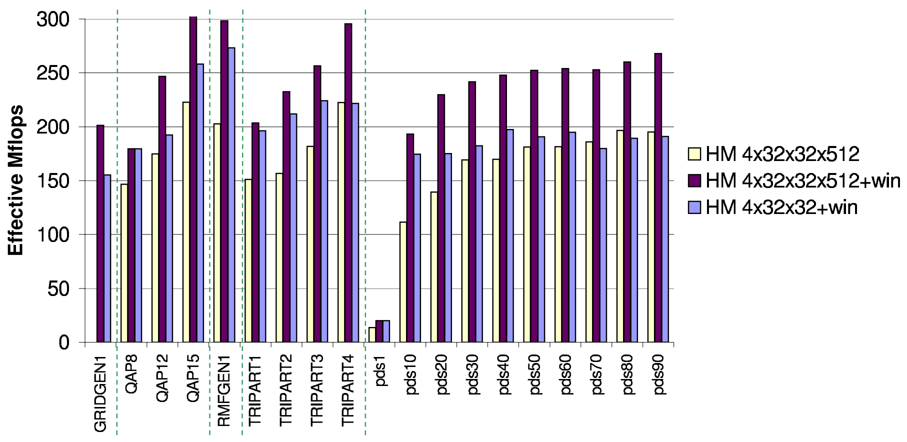


Fig. 5. HM performance for several input matrices

Next, we present results obtained by four different sparse Cholesky factorization codes. Figure 6 shows the results obtained with each of them for the set of matrices introduced above. Matrix families are separated by dashed lines.

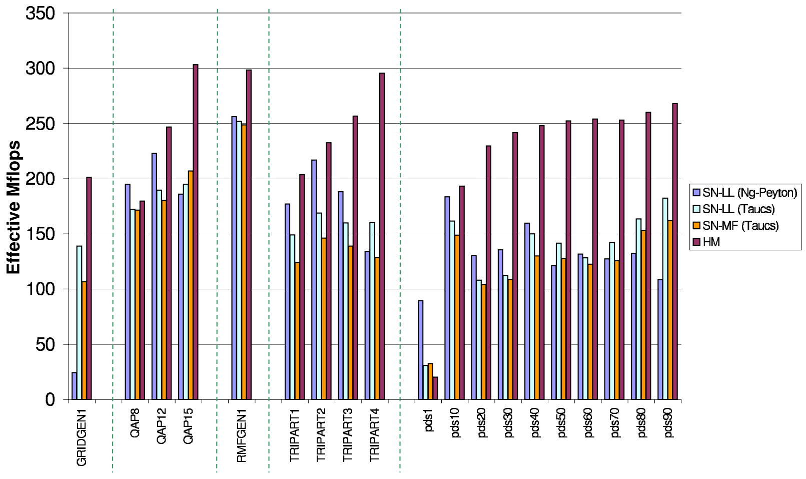


Fig. 6. Performance of several sparse Cholesky factorization codes

The first bar corresponds to a supernodal left-looking block Cholesky factorization (SN-LL (Ng-Peyton)) [20]. It takes as input parameters the cache size and unroll factor desired. This algorithm performs a 1D partitioning of the matrix. A supernode can be split into *panels* so that each panel fits in cache. This code has been widely used in several packages such as LIPSOL [24], PCx [7], IPM [6] or SparseM [15]. Although the test matrices we have used are in most cases very sparse, the number of elements per column is in some cases large enough so that a few columns fill the first level cache. Thus, a one-dimensional partition of the input matrix produces poor results. As the problem size gets larger, performance degrades heavily. We noticed that, in some cases, we could improve results by specifying cache sizes multiple of the actual first level cache. However, performance degrades in all cases for large matrices.

The second and third bars correspond to sequential versions of the supernodal left-looking (SN-LL) and supernodal multifrontal (SN-MF) codes in the TAUCS package (version 2.2) [13]. In these codes the matrix is represented as a set of supernodes. The dense blocks within the supernodes are stored in a recursive data layout matching the dense block operations. The performance obtained by these codes is quite uniform.

Finally, the fourth bar shows the performance obtained by our right looking sparse hypermatrix Cholesky code (HM). We have used windows within data submatrices and SML [12] routines to improve our sparse matrix application based on hypermatrices. A fixed partitioning of the matrix has been used. We present results obtained for data submatrix sizes 4×32 and upper hypermatrix levels with sizes 32×32 and 512×512.

We have included matrix pds1 to show that for small matrices the hypermatrix approach is usually very inefficient. This is due to the large overhead introduced by blocks

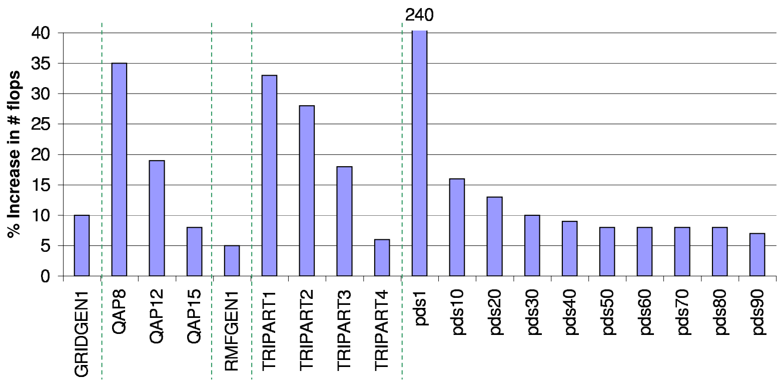


Fig. 7. Increase in number of operations in sparse HM Cholesky (4x32 + windows)

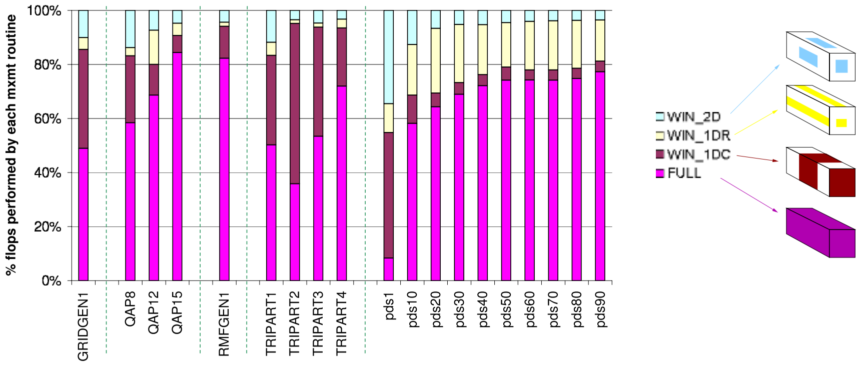


Fig. 8. HM flops per MxMt subroutine type

which have many zeros. Figure 7 shows the percentage increase in number of flops in sparse HM Cholesky w.r.t. the minimum (using a Compact Sparse Row storage). For large matrices however, blocks are quite dense and the overhead is much lower. Performance of HM Cholesky is then better than that obtained by the other algorithms tested. Note that the hypermatrix Cholesky factorization usually improves its performance as the problem size gets larger. There are two reasons for this. One is the better usage of the memory hierarchy: locality is properly exploited with the two dimensional partitioning of the matrix which is done in a recursive way using the HM structure. The other is that since blocks tend to be larger, more operations are performed by efficient $M \times M^t$ routines.

Figure 8 shows the percentage of multiplication operations performed by each of the four $M \times M^t$ routines we use. We focus on matrix multiplication because it is, by far, the most expensive operation within the Cholesky factorization.

FULL refers to the routine where all matrix dimensions are fixed. This is the most efficient of the four routines. WIN_1DC names the routine where windows are used for columns while WIN_1DR indicates the equivalent applied to rows. Finally, WIN_2D denotes the case where windows are used for both columns and rows. Thus, for the latter, no dimensions are fixed at compilation time and it becomes the least efficient of all four routines. WIN_1DC computes matrix multiplications more efficiently than WIN_1DR because of the constant leading dimension used for all three matrices. This is the reason why the performance obtained for matrices in the TRIPARTITE set is better than that obtained for matrices with similar size belonging to the PDS family. The performance obtained for the largest matrix in our test suite, namely matrix GRIDGEN1, is less than half of the theoretical peak for the machine used. This is basically due to the dispersion of data in this matrix which leads to the usage of the FULL $M \times M^t$ routine less than 50% of the time. In addition, the least efficient routine WIN_2D is used to compute about 10% of the operations.

4 Future Work

The results presented here have been obtained by splitting the matrix into blocks of fixed size for each hypermatrix level. However, we can also create the hypermatrix structure using the information from the supernodal elimination tree. The results for low amalgamation values are very poor and have not been presented. However, we expect an improvement in the performance of our sparse hypermatrix Cholesky code with different (larger) amalgamation values. In the near future we will experiment with different amalgamation values and algorithms.

We would also like to improve our code by avoiding the operation on very small matrices. We are currently studying the way to extend our code so that it can work with supernodes in addition to dense matrices in the last level of the hypermatrix (data submatrix level).

Since our approach seems promising for large matrices we want to extend it to work out-of-core. We think that the hypermatrix scheme also provides a good platform for exploiting the higher levels of the memory hierarchy.

Finally, we plan to use our code as the basis for a parallel implementation of the sparse Cholesky factorization.

5 Conclusions

A two dimensional partitioning of the matrix is necessary for large sparse matrices. The overhead introduced by storing zeros within dense data blocks in the hypermatrix scheme can be reduced by keeping information about a dense subset (window) within each data submatrix. Although some overhead still remains, the performance of our sparse hypermatrix Cholesky can be up to an order of magnitude better than that of a 1D supernodal block Cholesky (which tries to use the cache memory properly by splitting supernodes into panels). It also outperforms sequential supernodal left-looking and supernodal multifrontal routines based on a recursive data layout of blocks within the supernodes. Using windows and SML routines our HM Cholesky often gets over half of the processor's peak performance for medium and large sparse matrices factored sequentially in-core.

References

1. C. Ashcraft and R. G. Grimes. The influence of relaxed supernode partitions on the multifrontal method. *ACM Trans. Math. Software*, 15:291–309, 1989.
2. M. Ast, C. Barrado, J.M. Cela, R. Fischer, O. Laborda, H. Manz, and U. Schulz. Sparse matrix structure for dynamic parallelisation efficiency. In *Euro-Par 2000,LNCS1900*, pages 519–526, September 2000.
3. M. Ast, R. Fischer, H. Manz, and U. Schulz. PERMAS: User's reference manual, INTES publication no. 450, rev.d, 1997.
4. T. Badics. RMFGEN generator., 1991. Code available from ftp://dimacs.rutgers.edu/pub/netflow/generators/network/genrmf.
5. W.J. Carolan, J.E. Hill, J.L. Kennington, S. Niemi, and S.J. Wichmann. An empirical evaluation of the KORBX algorithms for military airlift applications. *Oper. Res.*, 38:240–248, 1990.
6. J. Castro. A specialized interior-point algorithm for multicommodity network flows. *SIAM Journal on Optimization*, 10(3):852–877, September 2000.
7. J. Czyzyk, S. Mehrotra, M. Wagner, and S. J. Wright. PCx User's Guide (Version 1.1). Technical Report OTC 96/01, Evanston, IL 60208–3119, USA, 1997.
8. I.S. Duff. Full matrix techniques in sparse Gaussian elimination. In *Numerical analysis (Dundee,1981). Lect. Notes in Math.*, 912:71–84. Springer, 1982.
9. A. Frangioni. Multicommodity Min Cost Flow problems. Data available from http://www.di.unipi.it/di/groups/optimize/Data/.
10. G.Von Fuchs, J.R. Roy, and E. Schrem. Hypermatrix solution of large sets of symmetric positive-definite linear equations. *Comp. Meth. Appl. Mech. Eng.*, 1:197–216, 1972.
11. A. Goldberg, J. Oldham, S. Plotkin, and C. Stein. An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow. In *IPCO*, 1998.
12. J.R. Herrero and J.J. Navarro. Improving Performance of Hypermatrix Cholesky Factorization. In *Euro-Par 2003,LNCS2790*, pages 461–469. Springer, August 2003.
13. D. Irony, G. Shklarski, and S. Toledo. Parallel and fully recursive multifrontal sparse Cholesky. In *ICCS 2002,LNCS2330*, pages 335–344. Springer, April 2002.
14. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. TR95-035, Dep. Comp. Sci., U. of Minnesota, Oct. 1995.
15. R. Koenker and P. Ng. SparseM: A Sparse Matrix Package for R, 2003. http://cran.r-project.org/src/contrib/PACKAGES.html#SparseM.
16. Y. Lee and J. Orlin. GRIDGEN generator., 1991. Code available from ftp://dimacs.rutgers.edu/pub/netflow/generators/network/gridgen.
17. J. H. W. Liu. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications*, 11(1):134–172, 1990.
18. J. W. Liu, E. G. Ng, and B. W. Peyton. On finding supernodes for sparse matrix computations. *SIAM J. Matrix Anal. Appl.*, 14(1):242–252, January 1993.
19. NetLib. Linear programming problems. http://www.netlib.org/lp/.
20. E.G. Ng and B.W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comput.*, 14(5):1034–1056, 1993.
21. A. Noor and S. Voigt. Hypermatrix scheme for the STAR–100 computer. *Computers and Structures*, 5:287–296, 1975.
22. E. Rothberg. Performance of panel and block approaches to sparse Cholesky factorization on the IPSC/860 and Paragon multicomputers. *SIAM J. Sci. Comput.*, 17(3):699–713, 1996.
23. R.C. Whaley and J.J. Dongarra. Automatically tuned linear algebra software. In *Supercomputing '98*, pages 211–217. IEEE Computer Society, Nov 1998.
24. Y. Zhang. Solving large–scale linear programs by interior–point methods under the MATLAB environment. TR 96–01, Baltimore, MD 21228–5398, USA, 1996.

Maximum-Weighted Matching Strategies and the Application to Symmetric Indefinite Systems*

Stefan Röllin[1] and Olaf Schenk[2]

[1] Integrated Systems Laboratory, Swiss Federal Institute of Technology Zurich
Gloriastrasse 35, CH-8092 Zurich, Switzerland
roellin@iis.ee.ethz.ch
http://www.iis.ee.ethz.ch

[2] Department of Computer Science, University Basel
Klingelbergstrasse 50, CH-4056 Basel, Switzerland
olaf.schenk@unibas.ch
http://www.informatik.unibas.ch/personen/schenk_o.html

Abstract. The problem of finding good numerical preprocessing methods for the solution of symmetric indefinite systems is considered. Special emphasis is put on symmetric maximum-weighted matching strategies. The aim is to permute large elements of the matrix to diagonal blocks. Several variants for the block sizes are examined and the accuracies of the solutions are compared. It is shown that maximum-weighted matchings improve the accuracy of sparse direct linear solvers. The use of a strategy called FULL CYCLES results in an accurate and reliable factorization. Numerical experiments validate these conclusions.

1 Introduction

It is now commonly known that maximum-weighted bipartite matching algorithms [5] – developed to place large entries on the diagonal using nonsymmetric permutations and scalings – greatly enhance the reliability of linear solvers [1,5] for nonsymmetric linear systems. The goal is to transform the coefficient matrix A with diagonal scaling matrices D_r and D_c and a permutation matrix P_r so as to obtain an equivalent system with a matrix $P_r D_r A D_c$ that has nonzeros on the diagonal and is better scaled and more diagonally dominant. This preprocessing has a beneficial impact on the accuracy of the solver and it also reduces the need for partial pivoting, thereby speeding up the factorization process. These maximum-weighted matchings are also applicable to symmetric indefinite systems, but the symmetry of the indefinite systems is lost and therefore, the factorization would be more expensive concerning both time and required memory. Nevertheless, modified weighted matchings can be used for these systems. The goal is to preserve symmetry and to define a permutation, such that the large elements of the permuted matrix lie on diagonal blocks (not necessarily on the diagonal as for nonsymmetric matrices). This technique based on numerical criterion was first presented by Gilbert and Duff [4] and further extended using a structural metric by Duff and Pralet [6].

* This work was supported by the Swiss Commission of Technology and Innovation under contract number 7036.1 ENS-ES, and the Strategic Excellence Positions on Computational Science and Engineering of the Swiss Federal Institute of Technology, Zurich.

We will use these matchings as an additional preprocessing method for the direct solver PARDISO [10]. This solver employs a combination of left- and right-looking Level 3 BLAS supernode techniques including a Bunch and Kaufman pivoting strategy restricted to a static structure of diagonal blocks corresponding to single supernodes.

2 Matching Strategies for Symmetric Indefinite Systems

We are looking for a symmetric permutation matrix P, such that the large elements of A lie in $p \times p$ diagonal blocks of the matrix PAP^T. The large elements do not necessarily lie on the diagonal as in the nonsymmetric case. As a first step, similar algorithms [5] as in the nonsymmetric case are used to compute the permutation matrix P_r as well as the diagonal scaling matrices D_r and D_c, which are used later to define a symmetric scaling matrix. The permutation corresponding to P_r can be broken up into disjoint cycles $c_1, \ldots, c_k$. The length of the cycles can be arbitrary but they sum up to the dimension of A. These cycles are used to define a symmetric permutation: $\sigma = (c_1, \ldots, c_k)$, which already results to the permutation matrix P, we are looking for. The matrix PAP^T will have k blocks of size $|c_1|, \ldots, |c_k|$ on the diagonal, if the adjacency structure of the nodes corresponding to a cycle are extended to a clique.

In the factorization, the rows and columns corresponding to a diagonal block are treated together. Large diagonal blocks will result in a large fill-in. A first possibility is to keep the k blocks and to put up with the fill-in. Another strategy is to break up long cycles of the permutation P_r to avoid large diagonal blocks. One has to distinguish between cycles of even and odd length. It is possible to break up even cycles into cycles of length two, without changing the weight of the matching due to the symmetry of the matrix. For each cycle, there are two possibilities to break it up. We use a structural metric [6] to decide which one to take. The same metric is also used for cycles of odd length, but the situation is slightly different. Cycles of length $2k + 1$ can be broken up into k cycles of length two and one cycle of length one. There are $2k + 1$ different possibilities to do this. The resulting 2×2 blocks will contain the matched elements. However, there is no guarantee that the diagonal element corresponding to the cycle of length one will be nonzero. Another possibility is therefore, to have $k - 1$ cycles of length two and one cycle of length three since there will be more freedom to choose an appropriate pivoting sequence in this case. Again, there are $2k + 1$ different ways to do the division.

As in the nonsymmetric case, it is possible to scale the matrix such that the absolute value of the matched elements is one and the other elements are equal or less than one in modulus. However, the original scalings cannot be used, since they are not symmetric, but they are used to define a symmetric scaling. The matrix $PD_sAD_sP^T$ will have the desired properties if we define a symmetric diagonal scaling matrix D_s as follows: $D_s = \sqrt{D_r D_c}$, where D_r and D_c are the diagonal matrices mentioned above.

In order to keep the fill-in small during the factorization, the matrix is reordered further with `MeTiS` [9] before carrying out the factorization. The block structure we have described before is kept in mind during this reordering step, such that the large elements still lie in diagonal blocks. We refer to the first strategy in [10, section 4] for more information on this topic.

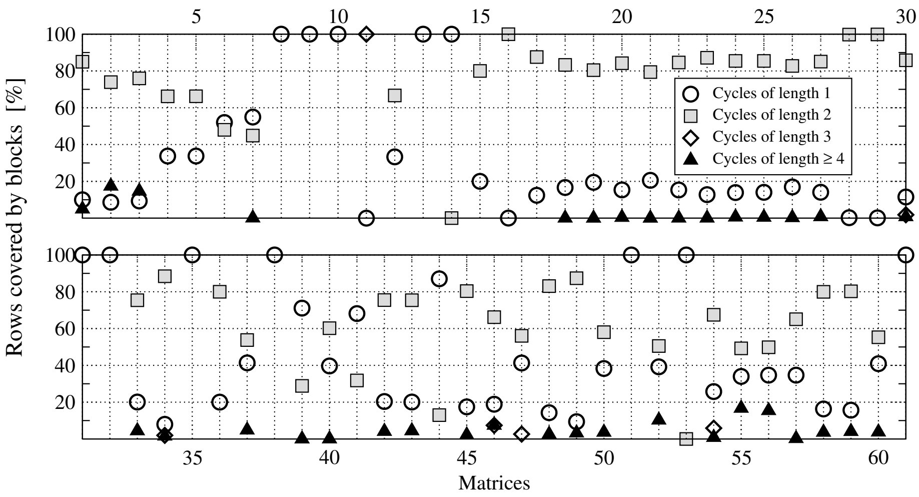


Fig. 1. Distribution of the cycle lengths for the tested matrices. Upper chart: matrices 1-30, lower chart: matrices 31-61

In the numerical experiments in section 3, we have used 61 sparse symmetric indefinite matrices. These matrices are described by Gould and Scott [7] and we have chosen the same ordering to list our results as they have. Figure 1 shows how long the cycles for the matrices are if the cycles are not broken up. In theory, they could be arbitrarily long. However, in practice, most of the cycles c_i are of length one or two. Only a small part of all cycles are longer than two. 40% of all matrices have only cycles of length one and two. In [4,6], the size of the diagonal blocks has been limited to one or two for the factorization process with MA57 [3]. Here, we extend these strategies and choose the following block sizes:

1. Do not break up cycles of length larger than two. This will probably result in more fill-in, but allows more choices to perform the pivoting within the diagonal blocks. Therefore, we expect to have better accuracy. This strategy will be called FULL CYCLES.
2. Divide blocks larger than two into blocks of size one and two, as described above. It is possible that the block of size one contains a zero element, which can lead to a zero pivot. However, contrary to [6], we do not permute this block to the end, but treat it in the same way as all other blocks. We use the name 2X2/1X1 for this choice.
3. For this possibility, an uneven cycle will be divided into blocks of size two and one block of size three. We name it 2X2/3X3.
4. Do not apply any kind of additional preprocessing based on symmetric matching. This strategy will be called DEFAULT.

In addition, all methods can be combined with a symmetric diagonal scaling based on $D_s = \sqrt{D_r D_c}$. Therefore, we can basically test eight strategies[3]. The difference between the eight methods is how the diagonal blocks are chosen and whether the matrix is scaled or not. In total, we have tested PARDISO with these eight different preprocessing strategies in combination with the 61 symmetric indefinite matrices.

All eight strategies are applied before performing the numerical factorization, for which we use the algorithm $\mathrm{LDL}^{\mathrm{T}} - \mathrm{SBK}$ described in [10]. This means that we use Bunch and Kaufman pivoting restricted to static data structures of diagonal supernode blocks in PARDISO. The coefficient matrix of the LDL^T factorization is perturbed whenever numerically acceptable pivots can not be found within a diagonal supernode block.

As stated above, it would be possible to apply the nonsymmetric permutations and scalings directly to the symmetric matrices, but one loses the symmetry for the factorization. Therefore, we have not tested a nonsymmetric permutation strategy. The strategy DEFAULT, SCALED deserves a special explanation. For this strategy, we do the preprocessing step only to compute the symmetric scaling matrix D_s and use it to scale the symmetric matrix. As a result, the largest absolute value in the matrix is one. This has an influence on the factorization, as we will see in the next section.

3 Numerical Experiments

For the computation of the symmetric permutation P and scaling D_s, we need to know a nonsymmetric permutation P_r and two diagonal scaling matrices D_r and D_c. We have used our own algorithm that is based on [8] for the numerical experiments to compute these matrices. We have also tested the algorithm MC64 from HSL (formerly known as Harwell Subroutine Library). The results are qualitatively the same and therefore we will only comment on the results of our algorithm.

The numerical results were conducted in sequential mode on a COMPAQ AlphaServer ES40 with eight GByte main memory and with four CPUs running at 667 MHz. A successful factorization does not mean that the computed solution is satisfactory, e.g., the triangular solves could be unstable due to numerical perturbation and thus give an inaccurate solution. With respect to this, we use two different accuracies to decide whether the factorization and solution process is considered to be successful:

1. **Standard Accuracy**: a factorization is considered successful if the factorization did not fail and the second scaled residual $\|Ax - b\|/(\|A\|\|x\| + \|b\|)$ that has been obtained after one step of iterative refinement is smaller than 10^{-8}.
2. **High Accuracy**: We say that the factorization has succeeded if the scaled residuals $\|Ax - b\|/(\|A\|\|x\| + \|b\|)$ are smaller than 10^{-14}. Furthermore, a factorization is considered to be successful if the residuals does not grow during the iterative refinement step.

The aim of the high accuracy is to see which strategy results in an accurate solution without the need of iterative refinements. Furthermore, we check whether the solution

[3] Please note that we will combine all strategies with a fill-in reduction method based on `MeTiS` that honors the special cycle structure.

Table 1. Number of solved matrices out of 61 for the eight different preprocessing strategies. The columns two to four show the number of systems, which are successfully solved, if standard accuracy or high accuracy is used

	Standard Accuracy of 10^{-8}	High Accuracy of 10^{-14}	
Iterative refinement	no	no	yes
DEFAULT	59	41	53
DEFAULT, SCALED	59	40	49
FULL CYCLES	59	52	54
FULL CYCLES, SCALED	61	49	51
2X2/1X1	57	47	52
2X2/1X1, SCALED	61	48	53
2X2/3X3	57	48	53
2X2/3X3, SCALED	61	47	52

process is stable by testing the convergence of the iterative refinement step. We compare the different strategies using the performance profiling system presented in [2] and which was also used in [7] to compare sparse direct solvers. Performance profiles are a convenient tool to compare a set of different algorithms $\mathcal{A}$ on a set of matrices $\mathcal{M}$. Each algorithm $i \in \mathcal{A}$ is run with each matrix $j \in \mathcal{M}$, which gives a statistic s_{ij}. The statistic s_{ij} might be the time for the factorization, the required memory or the norm of the scaled residuals. We assume that the smaller s_{ij} is, the better an algorithm performs. In case of a failure of the algorithm, we set $s_{ij} = \infty$. The performance profile $p_i(\alpha)$ of algorithm $i \in \mathcal{A}$ is defined as

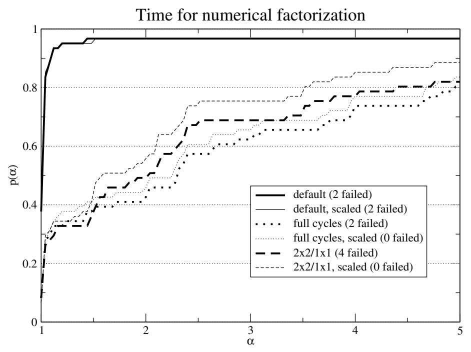
$$p_i(\alpha) := \frac{|\{j \in \mathcal{M} : s_{ij} \leq \alpha \hat{s}_j\}|}{|\mathcal{M}|},$$

where $\hat{s}_j := \min_{i \in \mathcal{A}}\{s_{ij}\}$ denotes the best statistics for matrix j. In other words, in a performance profile, the values on the y-axis indicate the fraction $p(\alpha)$ of all examples, which can be solved within α times the best algorithm.

3.1 Standard Accuracy of the Scaled Residuals

From Table 1 we see that the symmetric strategies (FULL CYCLES, 2X2/1X1, 2X2/3X3) with scaling successfully solve all matrices with an accuracy smaller than 10^{-8}. Without a scaling, slightly fewer systems can be solved. The memory requirements, the time to compute the factorization, and the accuracy varies between the eight methods. We describe the differences in the following.

Figure 2 shows the CPU time profile for the tested strategies, i.e. the time for the numerical factorization as well as the time for the overall solution. For the sake of clarity, we do not show the results for both 2X2/3X3 strategies, since there are only minor differences between them and the results of the strategies 2X2/1X1. The reason is



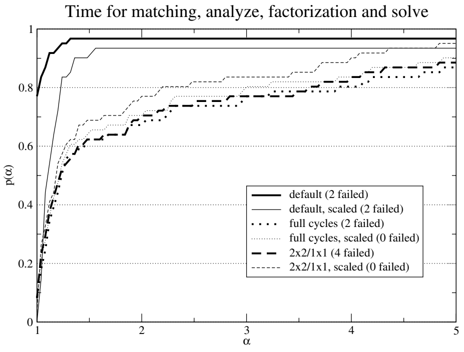
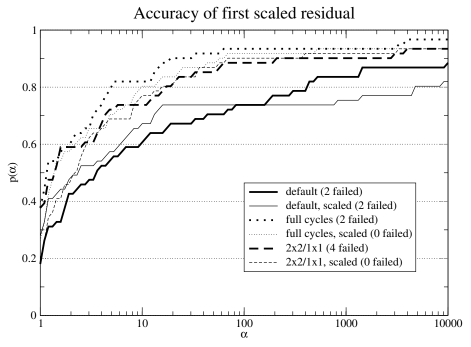


Fig. 2. Standard accuracy performance profile for the CPU time of the numerical factorization, and the complete solution including matching, analysis, factorization and solve



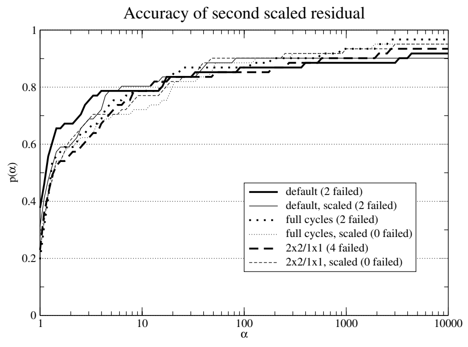


Fig. 3. Standard accuracy performance profile of the solution before and after one step of iterative refinement. The profile shows the results for the first and second scaled residual up to $\alpha = 10^5$

that these strategies only treat matrices differently that have odd cycles and only seven matrices have such cycles. The results for these matrices in combination with 2X2/1X1 and 2X2/3X3 differ only very slightly except for one matrix (Matrix 11: BLOCKQP1). The difference for this matrix is obvious, since it has almost only cycles of size three.

The default symmetric version of PARDISO appears to be the most effective solver. It can solve all but two systems, it has the smallest factorization time and the smallest total solution time for the complete solution. Comparing the profiles it can also be seen that the factorization time for the FULL CYCLES methods are the slowest of all strategies. This configuration requires the most memory since columns with different row structure are coupled together resulting in larger fill-in during factorization. Although 2X2/1X1 uses less memory and is faster than FULL CYCLES, it is still slower than both default strategies.

Figure 3 compares the first and second scaled residual that has been obtained after one step of iterative refinement. Here, the benefit of the preprocessing strategies becomes obvious. The most accurate solution before iterative refinement is provided by the strategy FULL CYCLES without scaling, followed closely by the other strategies.

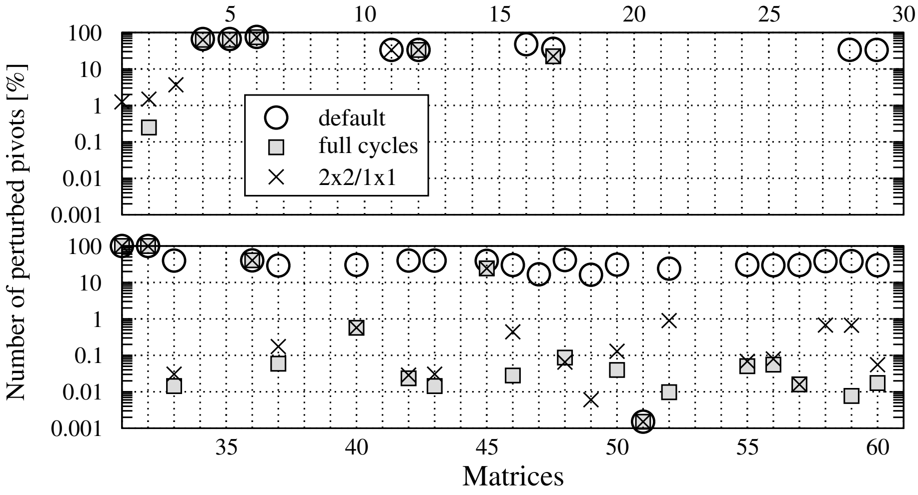


Fig. 4. Number of perturbed pivots during factorization for the tested matrices. Note the logarithmic scaling of the y-axis. If there is no symbol for a matrix, this means no perturbed pivots occurred during the factorization

Interestingly, the accuracy of the strategies with a scaling is slightly worse than without a scaling, even though the former can solve more problems. The DEFAULT strategy gives the least accurate results. However, after one step of iterative refinement, the accuracy of all methods is of the same order. It can be concluded that if the factorization time is of primary concern and if one step of iterative refinement can be applied, then the strategy DEFAULT is superior due to the fact that the accuracy of the scaled residuals is in the same order.

3.2 High Accuracy of the Scaled Residuals

Up to now, the DEFAULT strategy of PARDISO seems to be the best choice for most of the matrices if the accuracy of the solution is not the first criterion. However, the situation changes if the requirements for the solution are highly tightened and high accuracy without iterative refinement is used, as can be seen from column 3 in Table 1. No strategy is able to solve all matrices. The strategy FULL CYCLES is the most successful strategy. More precisely, for 52 matrices, the first scaled residual is smaller than 10^{-14}. The other preprocessing strategies are only slightly inferior. The results for both default strategies are much worse: PARDISO with the DEFAULT does not meet the accuracy requirements for 20 matrices and DEFAULT, SCALED fails for 21 matrices. That the use of blocks is beneficial is not surprising if the number of perturbed pivots during the factorization is considered, as depicted in Figure 4. In general, this number decreases by using a strategy with blocks on the diagonal. Especially for those matrices which are not successfully solved by the DEFAULT strategy, the number of perturbed pivots will decrease significantly resulting in a high accuracy of the solution without iterative refinement by using a FULL CYCLES or 2X2/1X1 strategy.

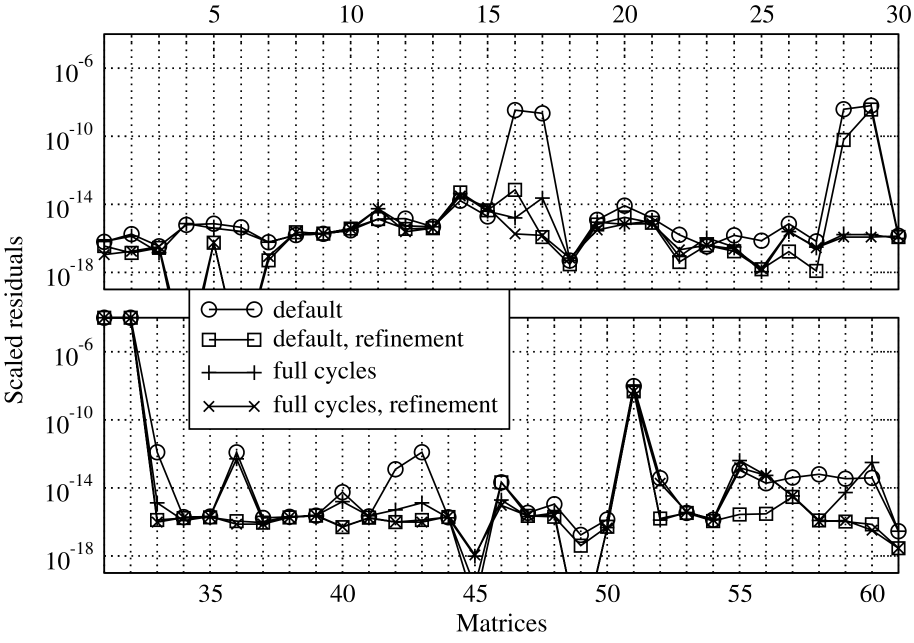


Fig. 5. Accuracy of the scaled residuals using DEFAULT and the FULL CYCLES strategy

The situation changes, if we look at the second scaled residuals, i.e. the residuals after one step of iterative refinement. The corresponding results are shown in the last column in Table 1. Although the FULL CYCLES strategy is still the best choice, there are only minor differences regarding the number of successful solves between the pre-processing strategies and the DEFAULT strategy. As a result, in can be concluded that the PARDISO default strategy with one step iterative refinement provides generally high accurate solutions and it can solve 53 out of 61 matrices with an accuracy of 10^{-14}.

In Figure 5, a comparison of the accuracy of the first and second scaled residual for all 61 matrices is given. This figure shows the accuracy for the DEFAULT and the FULL CYCLES strategy before and after one step of iterative refinement. It is clearly visible that the accuracy of the DEFAULT strategy with one step iterative refinement is a method that produces high accurate solutions for almost all matrices.

In Figures 6 and 7, the performance profiles are given for the total CPU time with and without one step of iterative refinement. Due to the fact that a high accurate solution is sought, there are now more failures for all strategies. As stated before, we set the statistic for the performance profile (e.g. the factorization time), to infinity in the case of a failure. Therefore, the performance profiles in Figures 2, 6 and 7 are different.

The best choice is FULL CYCLES without scaling if the accuracy is important and no iterative refinement can be made. The strategy 2X2/1X1 is only slightly worse, but on the average requires less memory for the factorization and solves the matrices faster. The fastest method that produces residuals with a similar accuracy is again the DEFAULT strategy using one step of iterative refinement.

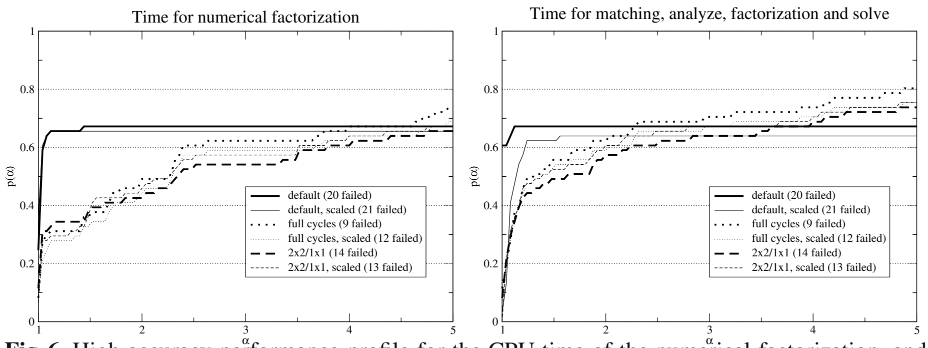


Fig. 6. High accuracy performance profile for the CPU time of the numerical factorization, and the time for the complete solution including matching, analysis, factorization and solve **without** iterative refinement

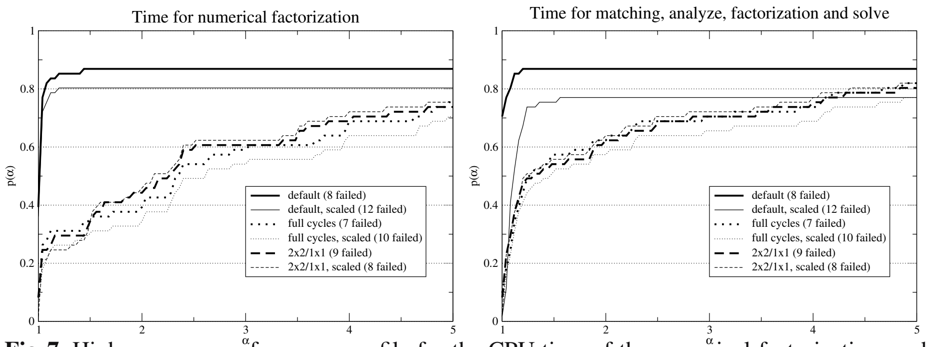


Fig. 7. High accuracy performance profile for the CPU time of the numerical factorization, and the time for the complete solution including matching, analysis, factorization and solve **with one step** of iterative refinement

4 Conclusion

We have investigated maximum-weighted matching strategies to improve the accuracy for the solution of symmetric indefinite systems. We believe that the current default option in PARDISO is a good general-purpose solver [10]. This is based on restricting pivoting interchanges within diagonal blocks corresponding to single supernodes. The factor LDL^T is perturbed whenever numerically acceptable pivots cannot be found within a diagonal block. The solution step is therefore performed with one step of iterative refinement. However, iterative refinement can be a serious problem in cases that involve a high number of solution steps. In these cases, matchings are advantageous as they improve the accuracy of the solution without performing iterative refinement. We have demonstrated that the matching strategies are very effective at the cost of a higher factorization time. Among the proposed strategies, FULL CYCLES is the most accurate one. Furthermore, our results also show that matchings strategies are more important than scalings and that these matching strategies improve the accuracy of the solution.

References

1. M. Benzi, J.C. Haws, and M. Tuma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Scientific Computing*, 22(4):1333–1353, 2000.
2. E. D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.
3. I. S. Duff. MA57 – a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30(2):118–144, June 2004.
4. I. S. Duff and J. R. Gilbert. Maximum-weighted matching and block pivoting for symmetric indefinite systems. In *Abstract book of Householder Symposium XV*, June 17-21, 2002.
5. I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Analysis and Applications*, 22(4):973–996, 2001.
6. I. S. Duff and S. Pralet. Strategies for scaling and pivoting for sparse symmetric indefinite problems. Technical Report TR/PA/04/59, CERFACS, Toulouse, France, 2004.
7. Nicholas I. M. Gould and Jennifer A. Scott. A numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations. *ACM Transactions on Mathematical Software*, 30(3):300 – 325, September 2004.
8. A. Gupta and L. Ying. On algorithms for finding maximum matchings in bipartite graphs. Technical Report RC 21576 (97320), IBM T. J. Watson Research Center, Yorktown Heights, NY, October 25, 1999.
9. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scientific Computing*, 20(1):359–392, 1998.
10. O. Schenk and K. Gärtner. On fast factorization pivoting methods for sparse symmetric indefinite systems. Technical Report CS-2004-004, Department of Computer Science, University of Basel, 2004. Submitted.

An Evaluation of Sparse Direct Symmetric Solvers: An Introduction and Preliminary Findings

Jennifer A. Scott[1], Yifan Hu[2], and Nicholas I.M. Gould[1]

[1] Computational Science and Engineering Department
Rutherford Appleton Laboratory
Chilton, Oxfordshire, OX11 0QX, England, UK
{n.i.m.gould,j.a.scott}@rl.ac.uk
[2] Wolfram Research, Inc., 100 Trade Center Drive
Champaign, IL61820, USA
yifanhu@wolfram.com

Abstract. In recent years a number of solvers for the direct solution of large sparse, symmetric linear systems of equations have been developed. These include solvers that are designed for the solution of positive-definite systems as well as those that are principally intended for solving indefinite problems. The available choice can make it difficult for users to know which solver is the most appropriate for their applications. In this study, we use performance profiles as a tool for evaluating and comparing the performance of serial sparse direct solvers on an extensive set of symmetric test problems taken from a range of practical applications.

1 Introduction

Solving linear systems of equations lies at the heart of numerous problems in computational science and engineering. In many cases, particularly when discretizing continuous problems, the system is large and the associated matrix A is sparse. Furthermore, for many applications, the matrix is symmetric; sometimes, such as in finite-element applications, A is positive definite, while in other cases, including constrained optimization and problems involving conservation laws, it is indefinite.

A direct method for solving a sparse linear system $Ax = b$ involves the explicit factorization of the system matrix A (or, more usually, a permutation of A) into the product of lower and upper triangular matrices L and U. In the symmetric case, for positive definite problems $U = L^T$ (Cholesky factorization) or, more generally, $U = DL^T$, where D is a block diagonal matrix with 1×1 and 2×2 blocks. Forward elimination followed by backward substitution completes the solution process for each given right-hand side b. Direct methods are important because of their generality and robustness. Indeed, for the 'tough' linear systems arising from some applications, they are currently the only feasible solution methods. In many other cases, direct methods are often the method of choice because the difficulties involved in finding and computing a good preconditioner for an iterative method can outweigh the cost of using a direct method. Furthermore, direct methods provide an effective means of solving multiple systems with the same A but different right-hand sides b because the factorization needs only to be performed once.

Since the early 1990s, many new algorithms and a number of new software packages for solving sparse symmetric systems have been developed. Because a potential user may be bewildered by such choice, our intention to compare the serial solvers (including serial versions of parallel solvers) on a significant set of large test examples from many different application areas. This study is an extension of a recent comparison by Gould and Scott [3] of sparse symmetric direct solvers in the mathematical software library HSL [7]. This earlier study concluded that the best general-purpose HSL package for solving sparse symmetric systems is currently MA57. Thus the only HSL direct solver included here is MA57, but for some classes of problems, other HSL codes may be more appropriate. For full details and results for the HSL symmetric solvers are given in [4].

The sparse solvers used in this study are listed in Table 1. The codes are discussed in detail in the forthcoming report [5]. Some of the solvers are freely available to academics while to use others it is necessary to purchase a licence. This information is provided in Table 2. For each code a webpage address is also given (or, if no webpage is currently available, an email contact is provided). Note that for non academic users, the conditions for obtaining and using a solver varies between the different packages.

Table 1. Solvers used in our numerical experiments. A '&' indicates both languages are used in the source code; 'F77/F90' indicates there is a F77 version and a F90 version

Code	Date/version	Language	Authors
BCSLIB-EXT	11.2001, v4.1	F77	The Boeing Company
MA57	01.2002, v1.0.0	F77/F90	I.S. Duff, HSL
MUMPS	11.2003, v4.3.2	F90	P.R. Amestoy, I.S. Duff, J.-Y. L'Excellent, and J. Koster
Oblio	12.2003, v0.7	C++	F. Dobrian and A. Pothen
PARDISO	02.2004	F77 & C	O. Schenk and K. Gärtner
SPOOLES	1999, v2.2	C	C. Ashcraft and R. Grimes
SPRSBLKLLT	1997, v0.5	F77	E.G. Ng and B.W. Peyton
TAUCS	08.2003, v2.2	C	S. Toledo
UMFPACK	04.2003, v4.1	C	T. Davis
WSMP	2003, v1.9.8	F90 & C	A. Gupta and M. Joshi, IBM

2 Test Environment

Our aim is to test the solvers on a wide range of problems from as many different application areas as possible. In collecting test data we imposed only two conditions:

- The matrix must be square and of order greater than $10,000$.
- The data must be available to other users.

Table 2. Academic availability of and contact details for the solvers used in our numerical experiments. * denotes source code is provided in the distribution

Code	Free to academics	Webpage / email contact
BCSLIB-EXT	×	www.boeing.com/phantom/BCSLIB-EXT/index.html
MA57*	×	www.cse.clrc.ac.uk/nag/hsl
MUMPS*	√	www.enseeiht.fr/lima/apo/MUMPS/
Oblio*	√	dobrian@cs.odu.edu or pothen@cs.odu.edu
PARDISO	√	www.computational.unibas.ch/cs/scicomp/software/pardiso
SPOOLES*	√	www.netlib.org/linalg/spooles/spooles.2.2.html
SPRSBLKLLT*	√	EGNg@lbl.gov
TAUCS*	√	www.cs.tau.ac.il/∼stoledo/taucs/
UMFPACK*	√	www.cise.ufl.edu/research/sparse/umfpack/
WSMP	√	www-users.cs.umn.edu/∼agupta/wsmp.html

The first condition was imposed because our interest is in large problems. The second condition was to ensure that our tests could be repeated by other users and, furthermore, it enables other software developers to test their codes on the same set of examples and thus to make comparisons with other solvers. Our test set comprises 88 positive-definite problems and 61 numerically indefinite problems. Numerical experimentation found 5 of the indefinite problems to be structurally singular and a number of others are highly-ill-conditioned. Any matrix for which we only have the sparsity pattern available is included in the positive-definite set and appropriate numerical values generated. Application areas represented by our test set include linear programming, structural engineering, computational fluid dynamics, acoustics, and financial modelling. A full list of the test problems together with a brief description of each is given in [6]. The problems are all available from

```
ftp://ftp.numerical.rl.ac.uk/pub/matrices/symmetric
```

In this study, performance profiles are used as a means to evaluate and compare the performance of the solvers on our set $\mathcal{T}$ of test problems. Let $\mathcal{S}$ represent the set of solvers that we wish to compare. Suppose that a given solver $i \in \mathcal{S}$ reports a statistic $s_{ij} \geq 0$ when run on example j from the test set $\mathcal{T}$, and that the smaller this statistic the better the solver is considered to be. For example, s_{ij} might be the CPU time required to solve problem j using solver i. For all problems $j \in \mathcal{T}$, we want to compare the performance of solver i with the performance of the best solver in the set $\mathcal{S}$.

For $j \in \mathcal{T}$, let $\hat{s}_j = \min\{s_{ij}; i \in \mathcal{S}\}$. Then for $\alpha \geq 1$ and each $i \in \mathcal{S}$ we define

$$k(s_{ij}, \hat{s}_j, \alpha) = \begin{cases} 1 \text{ if } \; s_{ij} \leq \alpha \hat{s}_j \\ 0 \text{ otherwise.} \end{cases}$$

The *performance profile* (see [1]) of solver i is then given by the function

$$p_i(\alpha) = \frac{\sum_{j \in \mathcal{T}} k(s_{ij}, \hat{s}_j, \alpha)}{|\mathcal{T}|}, \quad \alpha \geq 1.$$

Thus $p_i(1)$ gives the fraction of the examples for which solver i is the most effective (according to the statistic s_{ij}), $p_i(2)$ gives the fraction for which it is within a factor of 2 of the best, and $\lim_{\alpha \longrightarrow \infty} p_i(\alpha)$ gives the fraction for which the algorithm succeeded.

In this study, the statistics used are the CPU times required to perform the different phases of the solver, the number of nonzero entries in the matrix factor, and the total memory used by the solver (but in this paper, limitations on space allow us only to present CPU timings). Since some of the solvers we are examining are specifically designed for positive-definite problems (and may be unreliable, or even fail, on indefinite ones), we present our findings for the positive-definite and indefinite cases separately. In our tests, default values are used for all control parameters.

3 Preliminary Results

The numerical results were obtained on a Compaq DS20 Alpha server with a pair of EV6 CPUs; in our experiments only a single processor with 3.6 GBytes of RAM was used. The codes were compiled with full optimisation; the vendor-supplied BLAS were used. All CPU reported times are in seconds. A CPU limit of 2 hours was imposed for each code on each problem; any code that had not completed after this time was recorded as having failed. The scaled residual

$$\|b - Ax\| / (\|A\| \|x\| + \|b\|)$$

of each computed solution was checked before and after one step of iterative refinement; a residual after iterative refinement greater than 0.0001 causes an error to be flagged. In the reported tests, the input matrix A was not scaled.

3.1 Positive Definite Problems

The reliability of all the solvers in the positive-definite case was generally high. Only problem audikw was not solved by any code, this example being one of the two largest – it is of order roughly 950 thousand, and involves some 39 million nonzeros; the solvers with no out-of-core facilities were not able to allocate sufficient memory while the CPU time limit was exceeded for the remaining solvers. We present the performance profile for the CPU time for the complete solution (that is, the CPU time for analysing, factorising and solving for a single right-hand side) for the solvers in Figure 1, with the profiles for the separate analyse, factorise, and solve times in Figures 2 to 4. We see that, with the exception of SPOOLES and UMFPACK (which is primarily designed for unsymmetric problems), there is little to choose between the solvers when comparing the complete solution time. Many of the solvers use the nested dissection ordering from the METIS package [8] to obtain the pivot sequence and so they record similar analyse times. The multiple minimum degree algorithm used by SPRSBLKLLT is notably faster while WSMP

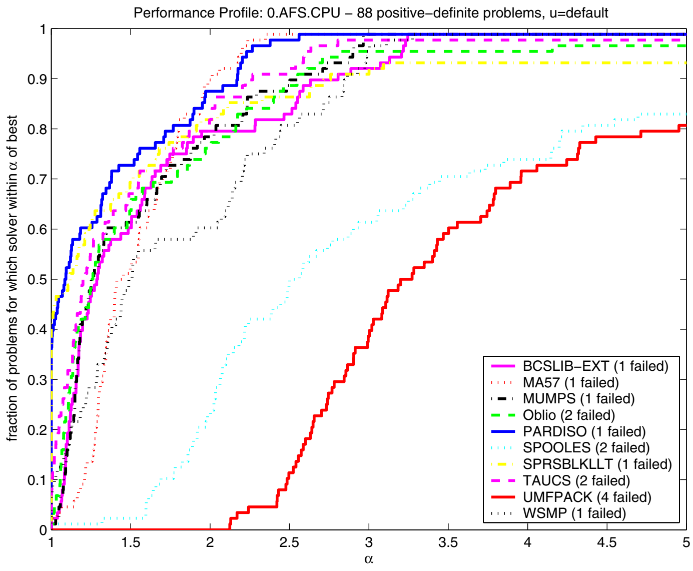


Fig. 1. Performance profile, $p(\alpha)$: CPU time (seconds) for the complete solution (positive-definite problems)

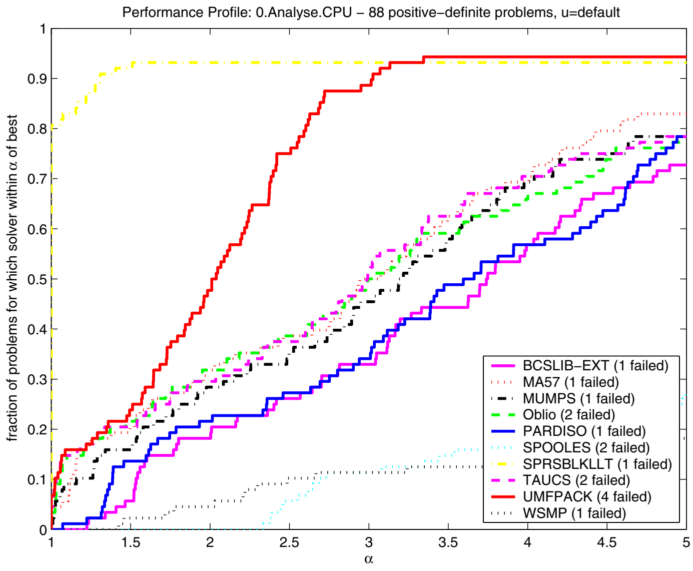


Fig. 2. Performance profile, $p(\alpha)$: CPU time (seconds) for analyse phase (positive-definite problems)

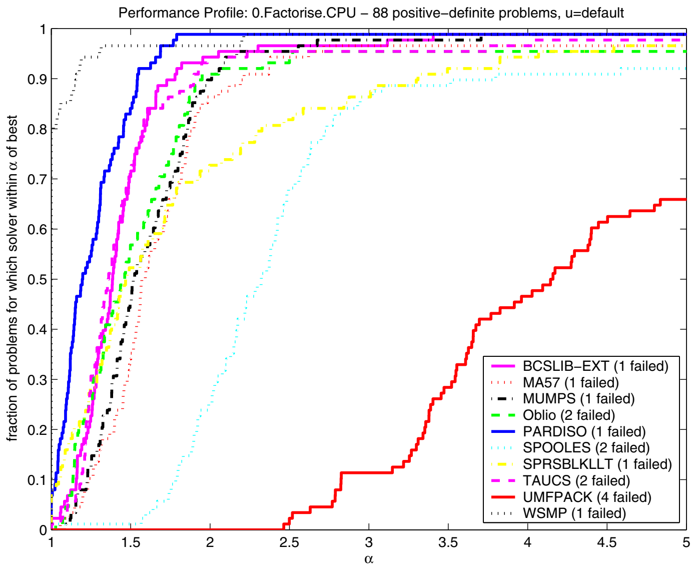


Fig. 3. Performance profile, $p(\alpha)$: CPU time (seconds) for the factorization (positive-definite problems)

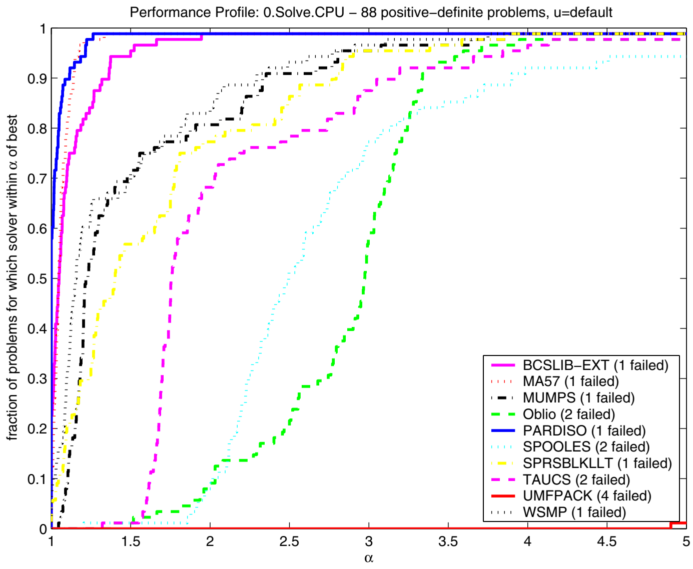


Fig. 4. Performance profile, $p(\alpha)$: CPU time (seconds) for the solve phase (positive-definite problems)

computes both a minimum local fill ordering and an ordering based on recursive bisection and selects the one that will result in the least fill-in. This extra investment pays dividends with WSMP having the fastest factorise times. In some applications, many solves may be required following the factorisation. The codes BCSLIB-EXT, MA57, and PARDISO have the fastest solve times.

3.2 Indefinite Problems

We now turn to the indefinite test suite. For these problems, pivoting is needed to maintain stability. The pivoting strategies offered by the codes are summarised in Table 3; more details are given in [5] and the references therein. Since SPRSBLKLLT and the tested version of TAUCS are designed only for solving definite problems, they are omitted. We have experienced problems when using SPOOLES for some indefinite systems, and thus results for SPOOLES are not currently included. MUMPS uses 1×1 pivots chosen from the diagonal and the factorization terminates if all the remaining (uneliminated) diagonal entries are zero. Because this may mean some of our test problems are not solved, at the authors' suggestion, we run both the symmetric and unsymmetric versions of MUMPS when testing indefinite examples.

Table 3. Default pivoting strategies

BCSLIB-EXT	Numerical pivoting with 1×1 and 2×2 pivots.
MA57	Numerical pivoting with 1×1 and 2×2 pivots.
MUMPS	Numerical pivoting with 1×1 pivots.
Oblio	Numerical pivoting with 1×1 and 2×2 pivots.
PARDISO	Supernode Bunch-Kaufmann within diagonal blocks.
SPOOLES	Fast Bunch-Parlett.
UMFPACK	Partial pivoting with preference for diagonal pivots.
WSMP	No pivoting.

The profiles for the indefinite results are given in Figures 5 to 8. The overall reliability of the solvers in the indefinite case was not as high as for the positive-definite one, with all the codes failing on some of the test problems. Because it does not include pivoting, WSMP had the highest number of failures. The majority of failures for the other codes were due to insufficient memory or the CPU time limit being exceeded or, in the case of symmetric MUMPS, no suitable diagonal pivots (singular matrices). The main exception was PARDISO, which had the fastest factorization times, but for two problems the computed solutions were found to be inaccurate.

We note that v1.0.0 of MA57 uses an approximate minimum degree ordering and computing this is significantly faster than computing the METIS ordering; this accounts for the fast MA57 analyse times. MA57 also has the fastest solve times; the solve times for PARDISO are slower even though it produces the sparsest factors since by default its solve phase includes one step of iterative refinement. Overall, PARDISO was the fastest code on our set of indefinite problems.

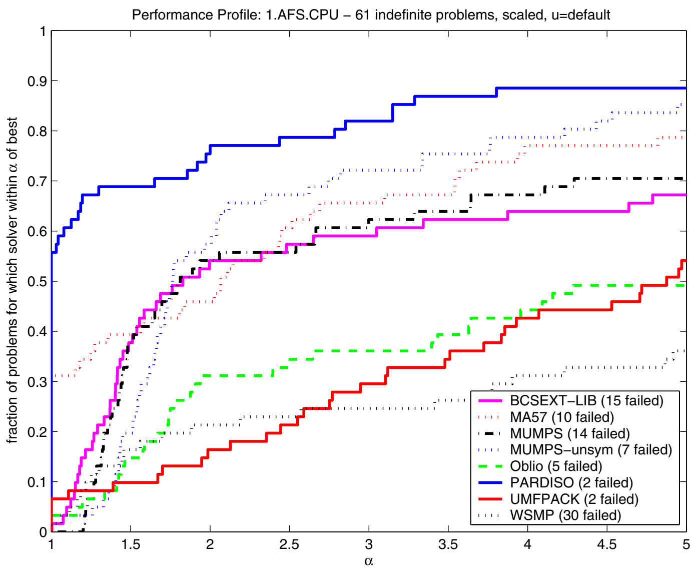


Fig. 5. Performance profile, $p(\alpha)$: CPU time (seconds) for the complete solution (indefinite problems)

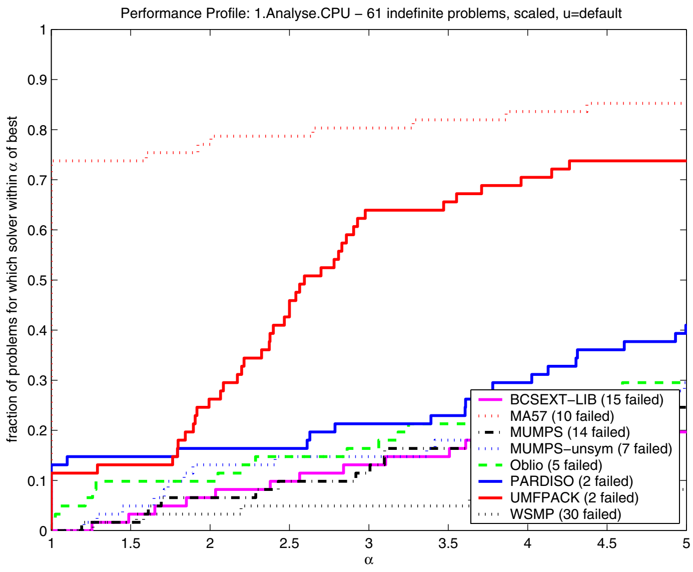


Fig. 6. Performance profile, $p(\alpha)$: CPU time (seconds) for analyse phase (indefinite problems)

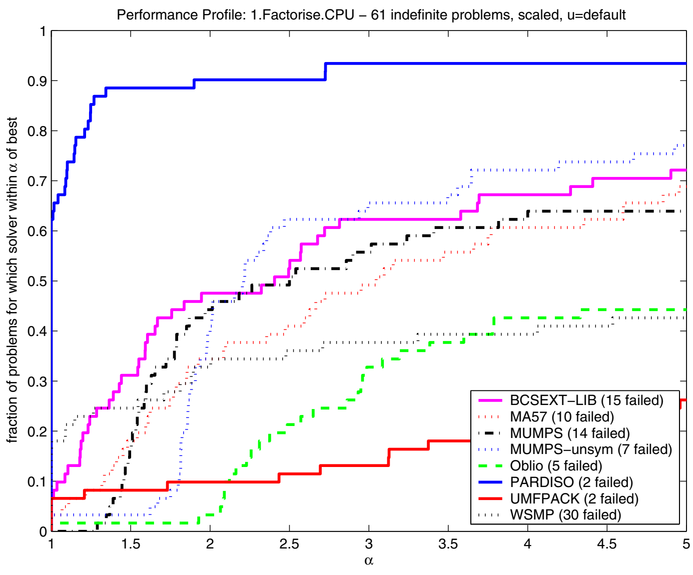


Fig. 7. Performance profile, $p(\alpha)$: CPU time (seconds) for the factorization (indefinite problems)

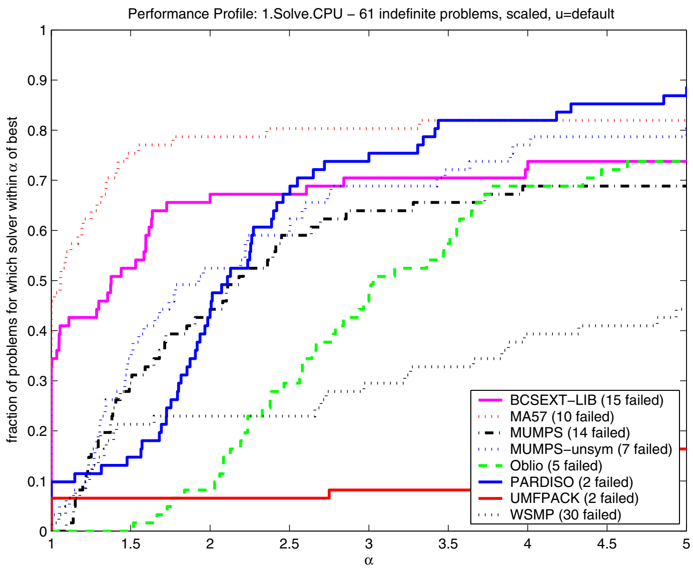


Fig. 8. Performance profile, $p(\alpha)$: CPU time (seconds) for the solve phase (indefinite problems)

4 General Remarks

In this paper, we have introduced our study of sparse direct solvers for symmetric linear systems and presented preliminary results. Full details of the study together with further information on all the solvers used will be given in the forthcoming report [5]. This report will also contain more detailed numerical results and analysis of the results. Furthermore, since performance profiles can only provide a global view of the solvers, the full results for all the solvers on each of the test problems will be made available in a further report [6]. We note that our preliminary findings have already lead to modifications to a number of the solvers (notably `MA57`, `PARDISO`, and `Oblio`), and to further investigations and research into ordering and pivoting strategies.

Acknowledgements

We are grateful to all the authors of the solvers who supplied us with copies of their codes and documentation, helped us to use their software and answered our many queries. Our thanks also to all those who supplied test problems.

References

1. E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, **91**(2), 201–213, 2002.
2. I.S. Duff, R.G. Grimes, and J.G. Lewis. Sparse matrix test problems. *ACM Trans. Mathematical Software*, **15**, 1–14, 1989.
3. N.I.M. Gould and J.A. Scott. A numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations. *ACM Trans. Mathematical Software*, **30**, 300-325, 2004.
4. N.I.M. Gould and J.A. Scott. Complete results for a numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations. Numerical Analysis Internal Report 2003-2, Rutherford Appleton Laboratory, 2003. Available from www.numerical.rl.ac.uk/reports/reports.shtml.
5. N.I.M. Gould, Y. Hu and J.A. Scott. A numerical evaluation of sparse direct solvers for the solution of large, sparse, symmetric linear systems of equations. Technical Report, Rutherford Appleton Laboratory, 2005. To appear.
6. N.I.M. Gould, Y. Hu, and J.A. Scott. Complete results for a numerical evaluation of sparse direct solvers for the solution of large, sparse, symmetric linear systems of equations. Numerical Snalysis Internal Report, Rutherford Appleton Laboratory, 2005. To appear.
7. HSL. A collection of Fortran codes for large-scale scientific computation, 2002. See http://hsl.rl.ac.uk/.
8. G. Karypis and V. Kumar. METIS: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices - version 4.0, 1998. See http://www-users.cs.umn.edu/ karypis/metis/

Treatment of Large Scientific Problems: An Introduction

Organizers: Zahari Zlatev[1] and Krassimir Georgiev[2]

[1] National Environmental Research Institute
Frederiksborgvej 399, P. O. Box 358, DK-4000 Roskilde, Denmark
[2] Institute for Parallel Processing, Bulgarian Academy of Sciences
Acad. G. Bonchev Str., Bl. 25-A, 1113 Sofia, Bulgaria

The computers are becoming faster and faster. Their capabilities to deal with very large data sets are steadily increasing. Problems that require a lot of computing time and rely on the use of huge files of input data can now be treated on powerful workstations and PCs (such problems had to be treated on powerful mainframes only 5-6 years ago). Therefore, it is necessary to answer the following two important questions:

– Are the computers that are available at present large enough?
– Do we need bigger (here and in the remaining part of this introduction, “a bigger computer" means a computer with larger memory discs, not a physically bigger computer) and faster computers for the treatment of the large-scale scientific problems, which appear in different fields of science and engineering and have to be resolved either now or in the near future?

These two questions can best be answered by using a quotation taken from a paper *“Ordering the universe: The role of mathematics"* written by Arthur Jaffe ([1]):

> *“Although the fastest computers can execute millions of operations in one second they are always too slow. This may seem a paradox, but the heart of the matter is: the bigger and better computers become, the larger are the problems scientists and engineers want to solve".*

The paper of Jaffe was published in 1984. The relevance of this statement can be illustrated by the following example. The difference between the requirements to the air pollution problems studied in 1984 and those studied at present can be expressed by the following figures about the sizes of the problems from this field that were solved 20 years ago and the problems treated now. At that time, i.e. in 1984, it was difficult to handle models containing, after the discretization, more than 2048 equations. One of the biggest problems solved at present, which can successfully be treated on large parallel computers, contains, again after the discretization, more than 160 000 000 ordinary differential equations. This system of ordinary differential equations had to be treated during about 250 000 time-steps. Moreover, many scenarios with this problem had to be handled in a comprehensive study related to future climate changes and high pollution levels. One of the major reasons for being able to handle such huge problems, about 80 000 times larger than the problems that were solved in 1984, is the availability of much bigger and much faster computers. There is no reason to assume that the development of bigger and faster computers will not continue. Therefore, the scientists could continue to plan the formulation and the treatment of bigger and bigger tasks, tasks which impose strong requirements for handling bigger and bigger data sets, because the solutions obtained in this way

- will be *closer* to the reality,
- will contain *more* details and *more useful* details

and

- will give *reliable* answers to questions which are at present open.

The main conclusion from the above short discussion can be formulated as follows. Although there are more and more problems which can successfully be handled (without any attempt to optimize the computational process) on workstations and PCs, it is still necessary, when the problems are very large, to run **highly optimized codes on big modern supercomputers**. The set of problems, which could be treated on workstations and PCs will become bigger and bigger in the future, because the computer power of the workstations and PCs will continue to be increasing. However, the requirements

- for more accuracy (i.e. the need to calculate more accurate and, thus, more reliable results)

and

- for obtaining more detailed information

are also steadily increasing. This means that the models are continuously becoming bigger and bigger. Of course, this is a well known, by the specialists, fact. In his paper [1] A. Jaffe stated (about 20 years ago) that the computers will always be too slow for the scientists that are dealing with very large applications. A. Jaffe

- was right,
- is still right

and

- will certainly continue to be right in the future.

In other words, there will always exist very advanced and very important for the modern society scientific problems which cannot be solved on workstations or PCs without imposing some non-physical assumptions during the development of the model. Thus, the fastest available computers will always be needed for the successful solution of the most important for the modern society tasks.

The exploitation of the new fast computers in the efforts to avoid non-physical assumptions and, thus, to develop and run more reliable and more robust large scientific models was the major topic of the special session on the *“Treatment of Large Scientific Models"*. In the papers, which were presented at this special session, the authors considered

- numerical methods which are both faster and more accurate,
- the use of parallel algorithms

and

- the organization of the computational process for efficient exploitation of the cache memory of the modern computer architectures.

The special session on the *“Treatment of Large Scientific Models"* was organized by Krassimir Georgiev (georgiev@parallel.bas.bg) and Zahari Zlatev (zz@dmu.dk).

Reference

1. Jaffe, A. (1984). *Ordering the universe: The role of mathematics*. SIAM Rev., 26: 475-488.

Towards a Parallel Multilevel Preconditioned Maxwell Eigensolver

Peter Arbenz[1], Martin Bečka[1,⋆], Roman Geus[2], and Ulrich Hetmaniuk[3,⋆]

[1] Institute of Computational Science, ETH Zürich, CH-8092 Zürich
[2] Paul Scherrer Institut, CH-5232 Villigen PSI
[3] Sandia National Laboratories, Albuquerque, NM 87185-1110, USA[⋆⋆]

Abstract. We report on a parallel implementation of the Jacobi–Davidson (JD) to compute a few eigenpairs of a large real symmetric generalized matrix eigenvalue problem

$$A\mathbf{x} = \lambda M\mathbf{x}, \qquad C^T\mathbf{x} = \mathbf{0}.$$

The eigenvalue problem stems from the design of cavities of particle accelerators. It is obtained by the finite element discretization of the time-harmonic Maxwell equation in weak form by a combination of Nédélec (edge) and Lagrange (node) elements.
We found the Jacobi–Davidson (JD) method to be a very effective solver provided that a good preconditioner is available for the correction equations that have to be solved in each step of the JD iteration. The preconditioner of our choice is a combination of a hierarchical basis preconditioner and a smoothed aggregation AMG preconditioner. It is close to optimal regarding iteration count and scales with regard to memory consumption.
The parallel code makes extensive use of the Trilinos software framework.

1 Introduction

Many applications in electromagnetics require the computation of some of the eigenpairs of the curl-curl operator,

$$\mathbf{curl}\,\mu_r^{-1}\mathbf{curl}\,\mathbf{e}(\mathbf{x}) - k_0^2\,\varepsilon_r\,\mathbf{e}(\mathbf{x}) = \mathbf{0}, \qquad \operatorname{div}\mathbf{e}(\mathbf{x}) = 0, \qquad \mathbf{x}\in\Omega, \tag{1.1}$$

in a bounded simply-connected, three-dimensional domain Ω with homogeneous boundary conditions $\mathbf{e}\times\mathbf{n} = \mathbf{0}$ posed on the connected boundary $\partial\Omega$. ε_r and μ_r are the relative permittivity and permeability. Equations (1.1) are obtained from the Maxwell equations after separation of the time and space variables and after elimination of the magnetic field intensity. While ε_r and μ_r are complex numbers in problems from waveguide or laser design, in simulations of accelerator cavities the materials can be

⋆ The work of these authors has been supported by the ETH research grant TH-1/02-4 on "Large Scale Eigenvalue Problems in Opto-Electronic Semiconductor Lasers and Accelerator Cavities".

⋆⋆ Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

assumed to be loss-free, thus admitting real ε_r and μ_r, whence all eigenvalues are real. Here, we will assume $\varepsilon_r = \mu_r = 1$. Thus, the discretization of (1.1) by finite elements leads to a real symmetric generalized matrix eigenvalue problem

$$A\mathbf{x} = \lambda M\mathbf{x}, \qquad C^T\mathbf{x} = \mathbf{0}, \tag{1.2}$$

where A is positive semidefinite and M is positive definite. In order to avoid spurious modes we approximate the electric field $\mathbf{e}$ by Nédélec (or edge) elements [12]. The Lagrange multiplier (a function) introduced to treat properly the divergence free condition is approximated by Lagrange (or nodal) finite elements [2].

In this paper we consider a parallel eigensolver for computing a few, i.e., five to ten of the smallest eigenvalues and corresponding eigenvectors of (1.2) as efficiently as possible with regard to execution time and consumption of memory space. In earlier studies [2] we found the Jacobi–Davidson algorithm [13,8] a very effective solver for this task. We have parallelized this solver in the framework of the Trilinos parallel solver environment [9].

In section 2 we review the symmetric Jacobi-Davidson eigensolver and the preconditioner that is needed for its efficient application. In section 3 we discuss data distribution and issues involving the use of Trilinos.

In section 4 we report on experiments that we conducted by means of problems originating in the design of the RF cavity of the 590 MeV ring cyclotron installed at the Paul Scherrer Institute (PSI) in Villigen, Switzerland. These experiments indicate that the implemented solution procedure is almost optimal in that the number of iteration steps until convergence only slightly depends on the problem size.

2 The Eigensolver

In this paper we focus on the symmetric Jacobi–Davidson algorithm (JDSYM) for solving (1.2). This algorithm is well-suited since it does not require the factorization of the matrices A or M. In [1,3,2] we found JDSYM to be the method of choice for this problem.

2.1 The Symmetric Jacobi–Davidson Algorithm

The Jacobi–Davidson algorithm has been introduced by Sleijpen and van der Vorst [13]. There are variants for all types of eigenvalue problems [4]. Here, we use a variant adapted to the generalized symmetric eigenvalue problem (1.2) as described in detail in [1,8].

Here we just sketch the algorithm. Let us assume that we have already computed q eigenvalues of (1.2) and have the corresponding eigenvectors available in the $n \times q$ matrix Q. Of course, $C^TQ = 0$. Let us further assume that we have available a search space $\mathcal{R}(V_k)$ with $V_k = [\mathbf{v}_1, \ldots, \mathbf{v}_k]$ with $C^TV_k = 0$ and Q^TMV_k = 0. The JDSYM proceeds in three steps to expand the search space by one dimension.

1. **Extraction.** In the extraction phase, a Ritz pair of (1.2) restricted to $\mathcal{R}(V_k)$ is computed. This amounts to computing the spectral decomposition of $V_k^TAV_k$ and selecting a particular eigenpair $(\tilde{\rho}, \tilde{\mathbf{q}})$ where $\tilde{\rho} = \rho(\tilde{\mathbf{q}})$ is the Ritz value in $\mathcal{R}(V_k)$

that best approximates the searched eigenvalue. Here, $\rho(\tilde{\mathbf{q}})$ denotes the Rayleigh quotient of $\tilde{\mathbf{q}}$.

2. **Correction.** In order to improve the actual best approximation $(\tilde{\rho}, \tilde{\mathbf{q}})$ a correction $\mathbf{t}$ is determined that satisfies the *correction equation*

$$\begin{aligned} &(I - M\widetilde{Q}\widetilde{Q}^T)(A - \tilde{\rho}M)(I - \widetilde{Q}\widetilde{Q}^T M)\mathbf{t} = -\mathbf{r}, \\ &\widetilde{Q}^T M\mathbf{t} = \mathbf{0}, \qquad C^T\mathbf{t} = \mathbf{0}, \qquad \widetilde{Q} = [Q, \tilde{\mathbf{q}}] \end{aligned} \tag{2.3}$$

where $\mathbf{r} = A\tilde{\mathbf{q}} - \tilde{\rho}\, M\tilde{\mathbf{q}}$ is called the residual at $\tilde{\mathbf{q}}$. $\mathbf{t}$ can be interpreted as a Newton correction at $\tilde{\mathbf{q}}$ for solving $A\mathbf{x} - \rho(\mathbf{x})M\mathbf{x} = \mathbf{0}$. For efficiency reasons the correction equation is solved only approximately by a Krylov subspace method.

3. **Extension.** The solution $\mathbf{t}$ of (2.3) is made M-orthogonal to V_k and orthogonal to C,

$$\hat{\mathbf{t}} = (I - V_k V_k^T M)(I - Y H^{-1} C^T)\mathbf{t}. \tag{2.4}$$

After M-normalization, $\hat{\mathbf{t}}$ is appended to V_k to yield V_{k+1}. Notice that $Y = M^{-1}C$ is a (very sparse) basis of the null space of A and that $H = Y^T C$ is the discretization of the Laplace operator in the nodal element space [2].

In order to limit the memory space the algorithm consumes the dimension of V_k is limited. If $\dim(V_k) = j_{\max}$ then the iteration is *restarted* meaning that the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{j_{\max}}$ are replaced by the $j_{\min}$ best Ritz vectors in V_k.

2.2 Solving the Correction Equation

For the Krylov subspace method to be efficient a preconditioner is a prerequisite. Following Fokkema et al. [6] for solving (2.3) we use preconditioners of the form

$$(I - M\widetilde{Q}\widetilde{Q}^T)K(I - \widetilde{Q}\widetilde{Q}^T M), \tag{2.5}$$

where K is a symmetric preconditioner of $A - \tilde{\rho}M$. For efficiency reasons we compute K only once for a fixed shift σ such that $K \approx A - \sigma M$. We experienced best results when σ is in the middle of the set of desired eigenvalues. However, in the experiments of this paper we choose σ close to but below the smallest eigenvalue we desire. This makes it possible to use the same K for all equations we are solving.

In each preconditioning step an equation of the form

$$(I - M\widetilde{Q}\widetilde{Q}^T)K\mathbf{c} = \mathbf{b} \quad \text{and} \quad \widetilde{Q}^T M\mathbf{c} = \mathbf{0}$$

has to be solved. The solution $\mathbf{c}$ is [8, p. 92]

$$\mathbf{c} = (I - K^{-1}M\widetilde{Q}(\widetilde{Q}^T M K^{-1} M\widetilde{Q})^{-1}\widetilde{Q}^T M)K^{-1}\mathbf{b}.$$

Briefly, the Krylov subspace method is invoked with the following arguments:

$$\begin{array}{lr} \text{system matrix:} & (I - M\widetilde{Q}\widetilde{Q}^T)(A - \tilde{\rho}M) \\ \text{preconditioner:} & (I - K^{-1}M\widetilde{Q}(\widetilde{Q}^T M K^{-T} M\widetilde{Q})^{-1}\widetilde{Q}^T M)K^{-1} \\ \text{right hand side:} & -(I - M\widetilde{Q}\widetilde{Q}^T)\mathbf{r} \\ \text{initial vector:} & \mathbf{0} \end{array} \tag{2.6}$$

Both the system matrix and the preconditioner are symmetric. However, because of the dynamic shift $\tilde{\rho}$ they can become indefinite. For this reason, the QMRS iterative solver [7] is suited particularly well.

2.3 The Preconditioner

Our preconditioner K, cf. (2.5), is a combination of a hierarchical basis preconditioner and an algebraic multigrid (AMG) preconditioner.

Since our finite element spaces consist of Nédélec and Lagrange finite elements of degree 2 and since we are using hierarchical bases we employ the hierarchical basis preconditioner that we used successfully in [2]. Numbering the linear before the quadratic degrees of freedom the matrices A and M in (1.2) get a 2-by-2 block structure,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \qquad M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix}.$$

Here, the $(1,1)$-blocks correspond to the bilinear forms involving linear basis functions. The hierarchical basis preconditioners as discussed by Bank [5] are stationary iteration methods for solving

$$\begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{pmatrix}, \qquad K_{ij} = A_{ij} - \sigma M_{ij}.$$

that respect the 2-by-2 block structure of A and M. If the underlying stationary method is the symmetric block Gauss–Seidel iteration then

$$K = \begin{bmatrix} K_{11} & \\ K_{21} & \widetilde{K}_{22} \end{bmatrix} \begin{bmatrix} K_{11} & \\ & \widetilde{K}_{22} \end{bmatrix}^{-1} \begin{bmatrix} K_{11} & K_{12} \\ & \widetilde{K}_{22} \end{bmatrix}.$$

The approximation $\widetilde{K}_{22}$ of K_{22} again represents a stationary iteration method. In a parallel environment the Jacobi iteration

$$\widetilde{K}_{22} = \operatorname{diag}(K_{22}) \tag{2.7}$$

is quite efficient and easy to implement.

For very large problems the direct solve with K_{11} becomes inefficient and in particular consumes far too much memory due to fill-in. In order to reduce the memory requirements of the two-level hierarchical basis preconditioner but at the same time not lose its optimality with respect to iteration count we replaced the direct solves by a single V-cycle of an AMG preconditioner. This makes our preconditioner a true multilevel preconditioner.

We found ML [11] the AMG solver of choice as it can handle unstructured systems that originate from the Maxwell equation discretized by linear Nédélec finite elements. ML implements a smoothed aggregation AMG method [15] that extends the non-smoothed aggregation approach of Reitzinger and Schöberl [10]. ML is part of Trilinos which is discussed in the next section.

3 Parallelization Issues

If the problems become very large then only the distribution of the data over a number of processors (and their memory) will make their solution feasible. Therefore, an efficient parallel implementation is prerequisite for being able to solve really large problems.

The parallelization of the algorithm requires proper data structures and data layout, parallel direct and iterative solvers and various preconditioners. We found the Trilinos Project [14] to be an efficient environment to develop such a complex parallel application.

3.1 Trilinos

The Trilinos Project is a huge effort to develop parallel solver algorithms and libraries within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific applications [14,9], still leveraging the value of established numerical libraries such as Aztec, the BLAS and LAPACK. Trilinos is a collection of compatible software packages that support parallel linear algebra computations, solution of linear and non-linear systems of equations, eigenvalue problems, and related capabilities. Trilinos is primarily written in C++, but part of the underlying code is implemented in C and Fortran.

Each Trilinos package is a self-contained, independent piece of software. The Trilinos fundamental layer, Epetra, provides a common look-and-feel and infrastructure. It implements basic objects like distributed vectors, matrices, graphs, linear operators and problems. It also implements abstract interfaces for Trilinos packages to interact with each other.

In our project we employ the package AztecOO, an object-oriented interface to the Aztec library of iterative solvers and preconditioners, and the package ML, that implements a smoothed aggregation AMG method capable of handling Maxwell equations. Direct solvers were managed by Amesos which is the Trilinos wrapper for various linear direct solvers.

To improve the original distribution of data the external Zoltan/ParMetis libraries were accessed through the EpetraExt interface. Some utilities from Teuchos and TriUtils packages were also used.

Trilinos offers effective facilities for parallel mathematical software developers. As it is still under development, the level of integration of the various packages into Trilinos and the quality of their documentation varies. E.g., at this time, ML for Maxwell equations is available only in the developer version of Trilinos.

3.2 Data Distribution

It is clear that a suitable data distribution can reduce communication expenses and balance computational load. The gain from such a redistribution can overcome the cost of this preprocessing step.

In the Trilinos context, distribution of matrices is by rows. A map defines which row goes on which processor. For the redistribution with Zoltan/ParMetis, an artificial graph G is constructed from portions of the matrices M, H, and C that contains a vertex for each node, edge, and face of the finite element mesh that participates at the computation.

Notice that neither M nor H works with all degrees of freedom. The Poisson matrix H has degrees of freedom located at vertices and on edges. The matrix M has degrees of freedom located on edges and element faces. The matrix C relates both. ParMetis tries to distribute the matrix such that (1) the number of nonzero elements per processor and thus the work load is balanced and (2) the number of edge cuts is minimal. The latter should minimize the communication overhead. In our experiments we observed significant decreases (up to 40%) of the execution time with the redistributed data.

4 Numerical Experiments

In this section we present experiments that we conducted by means of problems originating in the design of the RF cavity of the 590 MeV ring cyclotron installed at the Paul Scherrer Institute (PSI) in Villigen, Switzerland. The characteristics of this problem that we indicate by `cop40k` is given in Table 1 where orders n and number of nonzeros nnz of both the shifted operator $A - \sigma M$ and the discrete Laplacian H are listed. The second, artificial test example is a rectangular box denoted `box170k`. The experiments have been conducted on up to 16 processors of a 32 dual-node PC cluster in dedicated mode. Each node has 2 AMD Athlon 1.4 GHz processors, 2 GB main memory, and 160 GB local disk. The nodes are connected by a Myrinet providing a communication bandwidth of 2000 Mbit/s. The system operates with Linux 2.4.20. For our experiments we used the developer version of Trilinos on top of MPICH 1.2.5.

Table 1. Matrix characteristics

grid	$n_{A-\sigma M}$	$nnz_{A-\sigma M}$	n_H	nnz_H
cop40k	231668	4811786	46288	1163834
box170k	1030518	20767052	209741	5447883

In Tables 2 and 3 timings are given for computing the 5 smallest positive eigenvalues and corresponding eigenvectors using JDSYM with the multilevel preconditioner introduced earlier. We set $j_{\text{min}} = 6$ and $j_{\text{max}} = 15$. An approximate eigenpair $(\rho, \mathbf{q})$ is considered converged if the norm of the residual $\mathbf{r} = A\mathbf{q} - \rho M\mathbf{q}$ satisfies

$$\| \mathbf{r} \|_2 \leq \varepsilon\, c(M) \; \| \mathbf{q} \|_M,$$

where $c(M)$ approximates $\lambda_{\text{min}(M)}^{1/2}$ [3]. Notice that $\| \mathbf{r} \|_{M^{-1}} \leq \| \mathbf{r} \|_2 / \lambda_{\text{min}(M)}^{1/2}$.

In Tables 2 and 3 the execution times t are given for various processor numbers p. Both problems were too large to be solved on a single processor. Efficiencies $E(p)$ are given with respect to the performance of the run with two and six processors. Notice that the speedup from four to six processors in Table 3 is superlinear due to memory effects.

t_{prec} and t_{proj} give the percentage of the time the solver spent applying the preconditioner and the projector, respectively. The preconditioner in (2.6) is described in detail in section 2.3. The projector (2.4) is applied only once per outer iteration. Solving with H amounts to solving a Poisson equation [2]. This is done iteratively with the preconditioned conjugate gradient method. A multilevel preconditioner was provided by ML

Table 2. Results for matrix cop40k

p	t [sec]	$E(p)$	t_{prec}	t_{proj}	n_{outer}	$n_{\text{inner}}^{\text{avg}}$
2	1241	1.00	38%	16%	55	19.38
4	637	0.97	37%	17%	54	19.24
6	458	0.90	39%	18%	54	19.69
8	330	0.94	39%	17%	53	19.53
10	266	0.93	39%	19%	52	19.17
12	240	0.86	41%	20%	54	19.61
14	211	0.84	42%	20%	55	19.36
16	186	0.83	44%	20%	54	19.17

Table 3. Results for matrix box170k

p	t [sec]	$E(p)$	t_{prec}	t_{proj}	n_{outer}	$n_{\text{inner}}^{\text{avg}}$
4	7720	—	28%	22%	54	22.39
6	2237	1.00	39%	23%	55	22.47
8	1744	0.96	38%	23%	55	23.51
10	1505	0.89	38%	25%	56	22.54
12	1224	0.91	38%	25%	54	22.02
14	1118	0.86	39%	24%	55	23.76
16	932	0.90	38%	25%	54	22.30

applied to H. The projector consumes a high fraction of the execution time. This can be explained by the high accuracy (residual norm reduction by a factor 10^{10}) we required of the solution vector in order to guarantee that it satisfies the constraints $C^T\mathbf{x} = \mathbf{0}$. The fraction of the overall solver time of both preconditioner and projector are practically constant or increasing very slowly.

$n_{\text{inner}}^{\text{avg}}$ is the average number of inner iterations per outer iteration. The total number of matrix-vector multiplications and applications of the preconditioner is approximately $n_{\text{outer}} \times n_{\text{inner}}^{\text{avg}}$. The number of average inner iterations $n_{\text{inner}}^{\text{avg}}$ is almost constant indicating that the preconditioner does not deteriorate as p increases. In fact the only difference among the preconditioners of different p is the number of aggregates that are formed by ML. To reduce the number of messages aggregates are not permit to cross processor boundaries.

5 Conclusions

In conclusion, the parallel algorithm shows a very satisfactory behavior. The efficiency of the parallelization does not get below 83 percent for 16 processors. It slowly decreases as the number of processors increases, but this is natural due to the growing communication-to-computation ratio.

The accuracy of the results was satisfactory. The computed eigenvectors were M-orthogonal and orthogonal to C to machine precision. The 2-norm of the residuals of the computed eigenpairs were below 10^{-9}.

In the next future we will incorporate the hierarchical basis preconditioner for solves with the Poisson matrix H in the projector (2.4). Presently, we apply ML to the whole of H. We will also experiment with more powerful approximations $\widetilde{K}_{22}$ of K_{22} than just its diagonal, cf. (2.7). In [2] we used symmetric Gauss-Seidel instead of Jacobi preconditioning. The Gauss-Seidel preconditioner does not parallelize well, though.

References

1. P. Arbenz and R. Geus. A comparison of solvers for large eigenvalue problems originating from Maxwell's equations. *Numer. Linear Algebra Appl.*, 6(1):3–16, 1999.
2. P. Arbenz and R. Geus. Multilevel preconditioners for solving eigenvalue problems occuring in the design of resonant cavities. To appear in "Applied Numerical Mathematics". Paper available from `doi:10.1016/j.apnum.2004.09.026`.
3. P. Arbenz, R. Geus, and S. Adam. Solving Maxwell eigenvalue problems for accelerating cavities. *Phys. Rev. ST Accel. Beams*, 4:022001, 2001. Paper available from `doi:10.1103/PhysRevSTAB.4.022001`.
4. Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, Philadelphia, PA, 2000.
5. R. E. Bank. Hierarchical bases and the finite element method. *Acta Numerica*, 5:1–43, 1996.
6. D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst. Jacobi–Davidson style QR and QZ algorithms for the partial reduction of matrix pencils. *SIAM J. Sci. Comput.*, 20(1):94–125, 1998.
7. R. Freund and N. M. Nachtigal. Software for simplified Lanczos and QMR algorithms. *Appl. Numer. Math.*, 19:319–341, 1995.
8. R. Geus. *The Jacobi–Davidson algorithm for solving large sparse symmetric eigenvalue problems*. PhD Thesis No. 14734, ETH Zürich, 2002. (Available at URL `http://e-collection.ethbib.ethz.ch/show?type=diss&nr=14734`).
9. M. Heroux, R. Bartlett, V. Howle, R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams. An overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories, August 2003. To appear in the ACM Trans. Math. Softw.
10. S. Reitzinger and J. Schöberl. An algebraic multigrid method for finite element discretizations with edge elements. *Numer. Linear Algebra Appl.*, 9(3):223–238, 2002.
11. M. Sala, J. Hu, and R. S. Tuminaro. ML 3.1 Smoothed Aggregation User's Guide. Tech. Report SAND2004-4819, Sandia National Laboratories, September 2004.
12. P. P. Silvester and R. L. Ferrari. *Finite Elements for Electrical Engineers*. Cambridge University Press, Cambridge, 3rd edition, 1996.
13. G. L. G. Sleijpen and H. A. van der Vorst. A Jacobi–Davidson iteration method for linear eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 17(2):401–425, 1996.
14. The Trilinos Project Home Page. `http://software.sandia.gov/trilinos/`.
15. P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid based on smoothed aggregation for second and fourth order problems. *Computing*, 56(3):179–196, 1996.

On Improvement of the Volcano Search and Optimization Strategy

Venansius Baryamureeba and John Ngubiri

Institute of Computer Science, Makerere University
P.O.Box 7062, Kampala, Uganda
{barya,ngubiri}@ics.mak.ac.ug

Abstract. The ever-increasing load on databases dictates that queries do not need to be processed one by one. Multi-query optimization seeks to optimize queries grouped in batches instead of one by one. Multi-query optimizers aim at identifying inter and intra query similarities to bring up sharing of common sub-expressions and hence saving computer resources like time, processor cycles and memory. Of course, the searching takes some resources but so long as the saved resources are more than those used, there is a global benefit. Since queries are random and from different sources, similarities are not guaranteed but since they are addressed to the same schema, it is likely. The search strategy need to be intelligent such that it continues only when there is a high probability of a sharing (hence resource saving) opportunity. We present a search strategy that assembles the queries in an order such that the benefits are high, that detects null sharing cases and therefore terminates the similar sub-expressions' search as well as removing sub-expressions which already exist else where so as to reduce subsequent searching procedures for a global advantage.

AMS Subject Classification: 68M20, 68P20, 68Q85

Keywords: Query processing, Multi-query optimization, Volcano optimizer.

1 Introduction

Databases are becoming central in the running of organizations. The load put on them is therefore increasing. This is mostly from simultaneous, complex and close to real time requests. Single-query optimizers such as the System R optimizer [7] express query plans as trees and process one query at ago. They can work quite well for databases receiving a low traffic of simple queries. As queries become complex and many, they put a bigger load on computer resources. The need to optimize them in batches while exploiting similarities among them becomes irresistible. In fact, it is the only feasible way out.

Multi-query optimizers [2],[3],[8],[10] represent queries as Directed Acyclic Graphs (DAGs). Others like in [9] use an AND-OR DAGs so as to ease extensibility. An AND-OR DAG is a DAG with two types of nodes:- the AND nodes and the OR nodes. AND nodes have only OR nodes as children and OR nodes have only AND nodes as children. An AND node represents a relational algebraic operation such as join ($\bowtie$), select (σ) and project(π). AND nodes are as well referred to as $operational$ nodes. OR

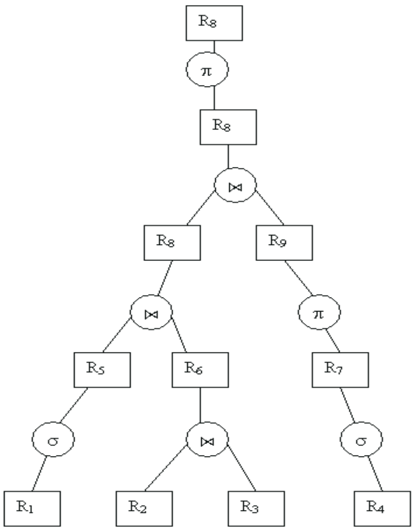


Fig. 1. Directed Acyclic Graph

nodes represents an equivalent logical expression of the child AND node and its children. They are as well referred to as *equivalence* nodes. Multi-query optimizers aim at traversing the different query plans(AND-OR DAGs) and identify sub-expressions (equivalent nodes) in which sharing is possible. Sharing of node outcomes saves memory, disc access and computation costs hence lowering the global resource requirements for the execution of the queries. Though searching for sub-expressions to be shared takes some resources (time and processor cycles). If a good search strategy is used, the savings exceed the searching cost hence a global advantage. Generally, there are three cases with a possibility of materialization and sharing. If W is an equivalent node of a plan, then sharing is possible when:-

(i) Nodes produce exactly similar results. e.g $s_1 = \sigma_{x=a}(W)$ and $s_2 = \sigma_{x=a}(W)$. In this case, only one is executed and the other one just uses the output of the executed node.
(ii) The output of one node is an exact subset of the other(s). For example, if we have nodes like $s_3 = \sigma_{x\leq 6}(W)$ and $s_4 = \sigma_{x\leq 10}(W)$, s_4 is executed and s_3 is derived from it i.e. $s_3 = \sigma_{x\leq 6}(s_4)$.
(iii) Nodes retrieve data from the same (intermediate) relation or view but on different instances of the outermost constraint. For example, if we have $s_5 = \sigma_{x=10}(W)$ and $s_6 = \sigma_{x=15}(W)$, then a different node $s_7 = \sigma_{x=10 \vee x=15}(W)$ is created so that the two are derived from it i.e. $s_5 = \sigma_{x=10}(s_7)$, and $s_6 = \sigma_{x=15}(s_7)$.

The efficiency of the multi-query optimizer does not depend on how aggressively common sub-expressions are looked for but rather on the search strategy [9]. Given that multi-query optimization takes place on many complex queries with many relations, comparing sub-expressions exhaustively leads to too many comparisons hence high comparison cost and time. Work of Kyuseok et al. [6], Kyuseok [5], Sellis and Gosh

[10], Cosar et al.[1] as well as Park and Segar [3] are exhaustive in nature hence costly. The searching cost may exceed the pre and post optimization resource trade off hence leading to no global benefit.
Graefe and McKenna [2] proposed the Volcano approach to multi-query optimization. Roy et al. [8] improved the search strategy proposed by Graefe and McKenna [2] while Roy et al. [9] proposed improvements in the algorithm in [2]. Kroger et al. [4] observes that the main aim of a multi-query optimizer is to come up with a plan as cost effective as possible in a time as short as possible. This research relies on the search strategy's ability to quickly identify shareable nodes and detect future null sharing cases so as to optimize the search effort and time for a global advantage.

We discuss related research in Section 2, the common sub-expressions search strategy in Section 3, the improved algorithm in Section 4 and give concluding remarks in Section 5.

2 Related Work

2.1 The Basic Volcano Algorithm

The basic Volcano algorithm was proposed by Graefe and McKenna [2]. It uses transformation rules to generate a DAG representation of the search space. For efficiency reasons, the basic Volcano algorithm starts from the root node and expands towards the leaves. It uses the upper bound of the node costs so as to come up with decisions whether to continue with the plan. This is so because the exact costs can only be got from the leaves. For any optimal sub plan got, it is **materialized** so that if found again, it is reused without re-optimizing and re-computing it. This saves costs.

2.2 The Volcano SH

The Volcano SH algorithm was proposed by Roy et al [9] as an improvement to the basic Volcano algorithm. It observes that not all sub-expressions are worth materialization. Sub-expressions are only materialized if materializing will cause global savings. It traverses the DAG bottom up and computes the cost of each equivalence node. If a node is found to exist more than once, a decision to materialize it is made. Materialization is only made if it will create a saving on the resources required to execute a query. If for example an equivalence node e has execution cost $= cost(e)$, Materialization cost $= matcost(e)$, reuse cost $= reusecost(e)$, and the node is used $numuses(e)$ times at run time. It is materialized if

$$cost(e)+matcost(e)+(reusecost(e)\times(numuses(e)-1)) < cost(e)\times numuses(e).$$

This simplifies to

$$reusecost(e) + \frac{matcost(e)}{numuses(e)-1} < cost(e)$$

[9] which is the materialization condition. Unless an equivalence node satisfies it, it is not cost effective to materialize it and therefore, it is not chosen for materialization. The Volcano SH starts from the leaves and advance towards the root. This makes it unable to

get the actual value of $numuses(e)$ for an equivalence node e since the number of times an equivalence node appears depends on the materialization status of the parents who, at the moment of reaching the equivalence node, are not yet met. The algorithm therefore uses an under estimate of $numuses(e)$ which is $numuses^{-}(e)$. $numuses^{-}(e)$ is got by simply counting the number of ancestors for the node in question [9]. The inputs are the basic Volcano optimal plans. It has a strengths that it avoids blind materialization. Materialization is only done when the algorithm is sure it will cause cost benefits. Therefore it is more efficient than the basic Volcano. It however has weaknesses, which include:-

(i) **Poor estimation of $numuses^{-}(e)$:** Estimating $numuses^{-}(e)$ for the node by counting the ancestors is inaccurate and can lead to wrong decisions. If for example node $\mathbf{a}$ is used twice to produce node $\mathbf{b}$ which is used thrice to produce node $\mathbf{c}$ ($\mathbf{c}$ has two parents and $\mathbf{c}$ together with the parents are all used once). The number of times $\mathbf{a}$ is used is 6. Using the under estimate in [9],we come up with 4. Since the under estimate is less than the actual value, it is okay. However, assuming all nodes were used once, then the number of times $\mathbf{a}$ is used is 1 yet the under estimate remains 4. This is dangerous since the node under estimate may permit materialization yet the actual value refuses it.In this case for example, since $\mathbf{a}$ appears only once, the decision to materialize it should not come up at all.

(ii) **Inter- query sharing and search optimization order:** The order of optimization in Volcano SH is immaterial [9]. This is because nodes in a plan are examined indipendently. Even the estimate of $numuses(e)$ is confined in the same plan [9]. This leads to excessive work being done. For example, in Figure 2, the roots of the two sub DAGs are exactly similar. Volcano SH reaches each of them via the leaves which is logically repetitive. Searching for the nodes top down, and then move horizontally across the plans would eliminate duplicate operations leading to similar outputs. Likewise, starting by a plan which share a lot with the rest would eliminate more operations hence smaller search space.

2.3 The Volcano RU

The Volcano RU was also proposed by Roy et al. in [9] and seek to exploit sharing beyond the optimal plans. If for example we have two optimal plans Q_1 =(A⋈B)⋈C and Q_2 =(A⋈C)⋈D, the Volcano SH algorithm would not permit sharing between Q_1 and Q_2 since no sub-expression is common between them. The Volcano RU however can adjust Q_1 to a locally sub optimal plan Q' =(A⋈C)⋈B such that a common sub-expression exists therefore sharing is possible.

In Volcano RU, queries are optimized one by one. For any sub-expression, the algorithm establishes whether it would cause savings if reused one more time. If it would, it is chosen for materialization for reuse in subsequent operations. Operations that follow are done putting into consideration that previous selected nodes are materialized. Volcano RU has a strength that it materializes beyond the optimal plans therefore more benefits are expected. It however has the following shortfalls that:

(i) **Over materialization:** Not all sub-expressions that would cause benefit when reused once actually exist in other queries in a batch. The criteria leads to over materialization hence resource wasting.

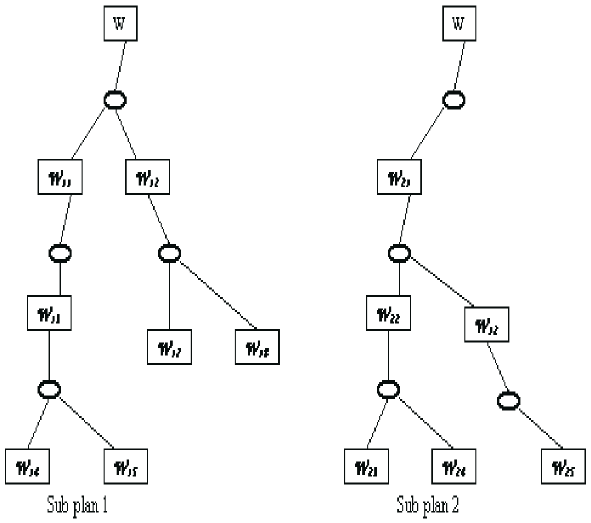


Fig. 2. Typical Sub-plans

(ii) **Order of optimization**: Though the order of optimization matters [9], the algorithm does no attempt to establish it.

3 Popularity and Inter-query Sharing Establishment

First, we establish the extent of sharing and summerize it in a **query sharing matrix**. The total number of equivalence nodes in a plan that have sharing partners in the batch is the **popularity** of the query while the number of relations with at least a relational operator, that make up a node is the node **order**. We chose an optimal processing order, eliminate duplicate operations as well as detecting no sharing opportunities. First, we create the sharing matrix and then base on it to optimize the batch.

3.1 The Greedy Search Algorithm

We get nodes from the Volcano best plans of the queries in the batch produced as in [2]. The greedy algorithm compares nodes pairwise to establish inter and intra sharing. M is accordingly updated. Using this approach, for a batch of n queries where the i^{th} query has k_i nodes, we need $\Sigma_{alli}k_i \times (\Sigma_{alli}k_i - 1)$ comparisons which is too much. We propose improvements:

1. **Eliminate duplicate searches:** The output of search Q_i and Q_j is equivalent to that between Q_j and Q_i. We therefore need to compare Q_i and Q_j only if $i \leq j$.
2. **Search by node order:** Sharing can only take place between nodes of the same order. We therefore need to group nodes by order and for a node, we search within the order.

3. **Null sharing prediction:** Since moving up the order make the nodes more specific, if there is no sharing for nodes of order m, then there is no sharing for nodes of order n where $n > m$. We therefore terminate higher nodes' search for the query pair.
4. **Zero order tip:** Relations (zero order node) are too broad to be considered for sharing. We only use them to find if any two queries are disjoint or not. If we get a sharing opportunity at this order, we leave M un updated and go to the next order. If however there are no nodes sharable, then the queries are disjoint.

3.2 The Improved Greedy Search Algorithm

We now present the greedy algorithm but enhanced such that the observations above are put into consideration in order to have a more optimal search. It outputs the sharing matrix.

```
for i = 1; i $\leq$ n; i++
    for j = i; j $\leq$ n; j++
        ORDER = 0
        repeat
            nextOrderPermision = false
            $S_i$ = set of nodes of order ORDER in query i
            node1 = first node in $S_i$
            $S_j$ = set of nodes of order ORDER in query j
            node2 = first node in $S_j$
            while($S_i$ still has nodes)
                while($S_j$ still has nodes)
                    if(node1 and node2 are sharable)
                        nextOrderPermision = true
                        if(ORDER = 0)
                            break out of the two while loops
                        else
                            increment $M[i, j]$ and $M[j, i]$
                            mark node1 and node2
                        endif
                    endif
                    node2 = next(node2)
                endwhile
                node1 = next(node1)
            endwhile
            ORDER = next(ORDER)
        until(nextOrderPermision = false or orders are exhausted)
    endfor
endfor
```

4 The Optimizing Algorithm

The new algorithm inputs the DAG made up of the basic Volcano plans for each query. The shareability information and popularities are got from M. In this algorithm plans are assigned focal roles in the order of their decreasing popularity, sharing for any focal plan is done in the stable marriage preference order of less popular plans, searching starts from a higher order and plans of Zero popularity are not assigned focal roles.

4.1 The Algorithm

```
S = set of plans that make up the virtual DAG in order of their popularity
focalPlan = first plan in S
repeat
    establish node cost and numuses for each node for the focal plan
    S* = subset of S who share at least a node with focalPlan query
    candidateOrder = highest order of focalPlan
    repeat
        e = node in candidateOrder
        if( e is marked)
            repeat
                traverse S* searching for marked equivalent nodes of the same order
                with and sharable with e
                increment numuses(e) whenever a sharable node is met
            until(S* is exhausted)
            if(sharable condition(e))
                chose which node to materialized and add it to the materializable
                set remove the rest
                update the DAG for the materialized node to cater for the removed
                nodes' parents un mark the chosen node
            endif
        else
            if(sharablecondition(e))
                add e to materialization node
            endif
        endif
    until(nodes in candidateOrder are exhausted)
    focalPlan = next(focalPlan)
until(plans on non zero popularity are exhausted)
```

4.2 Benefits of the New Algorithm

1. **Better estimation of $numuses(e)$:** The algorithm uses the exact number of times the node exists in the focal plan and increments by one whenever a sharable node found outside the focal plan. The nodes may actually be greater if the none focal plan nodes are used multiple times therefore $numuses^{-}(e) \leq numuses(e)\ \forall e$
2. **Elimination of null searches:** The sharing matrix has details of the extent of sharing for any pair of plans. if the entry for a pair is Zero, then we need not to search for shareability between them. If the popularity of a certain query is zero, then its plan is not involved in the search.
3. **Elimination of catered for optimizations:** If we have say three nodes of order five and they are sharable, and it is decided that one has to be materialized and the rest of the plans use it, then it is not worthwhile to process the other nodes since the ultimate goal is to get the root of the tree. This algorithm removes the sub DAGs

whose output can be got from materialized nodes so that such children do not enter the optimization process yet their output is catered for.

4. **Optimal order of optimization:** Since the strategy eliminates catered for sub-DAGs, its better if it does it as early as possible so that the subsequent search space is reduced without affecting the outcome. starting with the most popular query does this. This saves time, Memory and processor cycles.

5 Conclusion

We have shown that by using bottom up approach at search and up - down approach at optimization we are able to only continue searching for sharable nodes when they exist. Still identification of a sharable node leads to the removal of all children whose output are already catered for by the sharing. This reduces the sample space therefore the subsequent search cost and time.

References

1. A. Cosar, E. Lim and J. Srivasta. Multiple query Optimization with depth-first branch and bond and dynamic query ordering. *International Conference on Information and Knowledge Management*, 2001.
2. G. Graefe and W.J. McKenna. Extensibility and search efficiency in the Volcano Optimizer generator. *Technical report CU-CS-91-563. University of Colorado*, 1991.
3. J. Park and A. Seger. Using common sub-expressions to optimize multiple queries. *Proceedings of the IEEE International Conference on Data Engineering*,1988.
4. J. Kroger, P. Stefan, and A.Heuer. Query optimization: On the ordering of Rules. *Research paper, Cost - and Rule based Optimization of object - oriented queries (CROQUE) Project* University of Restock and University of Hamburg - Germany, 2001.
5. Kyuseok Shim. Advanced query optimization techniques for relational database systems. *PhD dissertation, University of Maryland*, 1993.
6. Kyuseok Shim, Timos.K. Sellis and Dana Nau. Improvements on a heuristic algorithm for multiple-query Optimization. *Technical report, University of Maryland, Department of Computer science*,1994.
7. P.G.Selinger, M.M Astrahan, D.D Chamberlin, R. A. Lorie and T.G Price: Access path selection in relational database management systems. *In preceeding of the ACM-SIGMOD Conference for the Management of Data*, 1979. pp 23-34.
8. Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh. Bhobe. Practical Algorithms for multi-query Optimization. *Technical Report, Indian Institute of Technology, Bombay*,1998.
9. Prasan Roy, S. Seshadri, S. Sudarshan, and Siddhesh Bhobe. Efficient and Extensible algorithms for Multi query optimization. *Research Paper, SIGMOD International Conference on management of data*,2001.
10. Timos. K. Sellis and S. Gosh. On Multi-query Optimization Problem. *IEEE Transactions on Knowledge and Data Engineering* 1990 pp. 262-266.

Aggregation-Based Multilevel Preconditioning of Non-conforming FEM Elasticity Problems

Radim Blaheta[1], Svetozar Margenov[2], and Maya Neytcheva[3]

[1] Institute of Geonics, Czech Academy of Sciences
Studentska 1768, 70800 Ostrava-Poruba, The Czech Republic
blaheta@ugn.cas.cz

[2] Institute for Parallel Processing, Bulgarian Academy of Sciences
Acad. G. Bonchev Str., Bl. 25A, 1113 Sofia, Bulgaria
margenov@parallel.bas.bg

[3] Department of Information Technology, Uppsala University
Box 337, SE-75105 Uppsala, Sweden
maya@it.uu.se

Abstract. Preconditioning techniques based on various multilevel extensions of two-level splittings of finite element (FE) spaces lead to iterative methods which have an optimal rate of convergence and computational complexity with respect to the number of degrees of freedom. This article deals with the construction of algebraic two-level and multilevel preconditioning algorithms for the Lamé equations of elasticity, which are discretized by Crouzeix-Raviart non-conforming linear finite elements on triangles. An important point to note is that in the non-conforming case the FE spaces corresponding to two successive levels of mesh refinements are not nested. To handle this, a proper aggregation-based two-level basis is considered, which enables us to fit the general framework of the two-level preconditioners and to generalize the method to the multilevel case. The derived estimate of the constant in the strengthened Cauchy-Bunyakowski-Schwarz (CBS) inequality is uniform with respect to both, mesh anisotropy and Poisson ratio, including the *almost incompressible case*.

1 Introduction

The target problem in this paper is the Lamé system of elasticity:

$$\sum_{j=1}^{2} \frac{\partial \sigma_{ij}}{\partial x_j} + f_i = 0, \qquad x \in \Omega, \qquad i = 1,2$$
$$u = 0, \qquad x \in \partial\Omega$$

where Ω is a polygonal domain in $\mathbb{R}^2$ and $\partial\Omega$ is the boundary of Ω. The stresses σ_{ij} and the strains ε_{ij} are defined by the classical Hooke's law, i.e.

$$\sigma_{ij}(u) = \lambda \left(\sum_{k=1}^{2} \varepsilon_{kk}(u) \right) \delta_{ij} + 2\mu\varepsilon_{ij}(u), \qquad \varepsilon_{ij}(u) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right).$$

We assume that the Lamé coefficients are piecewise constant in the domain Ω. The unknowns of the problem are the displacements $\mathbf{u}^t = (u_1, u_2)$. A generalization to nonhomogeneous displacement boundary condition is straightforward.

The Lamé coefficients are given by $\lambda = \dfrac{\nu E}{(1+\nu)(1-2\nu)}$, $\mu = \dfrac{E}{2(1+\nu)}$, where E stands for the elasticity modulus, and $\nu \in (0, \frac{1}{2})$ is the Poisson ratio. We use the notion *almost incompressible* for the case $\nu = \frac{1}{2} - \delta$ ($\delta > 0$ is a small parameter). Note that the boundary value problem becomes ill-posed when $\nu = \frac{1}{2}$ (the material is *incompressible*).

We assume that the domain Ω is discretized using triangular elements. The partitioning is denoted by $\mathcal{T}_\ell$ and is assumed to be obtained by ℓ regular refinement steps of a given coarse triangulation $\mathcal{T}_0$.
For $\mathbf{f} \in (L^2(\Omega))^2$, the weak formulation of the boundary value problem reads:
find $\mathbf{u} \in \mathcal{V} = \{\mathbf{v} \in (H^1(\Omega))^2, \mathbf{v}\mid_{\partial\Omega} = 0\}$ such that

$$a(\mathbf{u}, \mathbf{v}) = -\int_\Omega \mathbf{f}^T \mathbf{v} dx + \int_{\Gamma_N} \mathbf{g}^T \mathbf{v}\, ds, \qquad \forall \mathbf{v} \in \mathcal{V}. \tag{1}$$

Here Γ_N is that part of $\partial\Omega$ where there are active surface forces. The bilinear form $a(\mathbf{u}, \mathbf{v})$ is of the form

$$a(\mathbf{u}, \mathbf{v}) = \int_\Omega [\lambda div(\mathbf{u}) div(\mathbf{v}) + 2\mu \sum_{i,j=1}^{2} \varepsilon_{ij}(\mathbf{u})\varepsilon_{ij}(\mathbf{v})] dx \tag{2}$$

$$= \int_\Omega \langle C d(\mathbf{u}), d(\mathbf{v})\rangle dx, \tag{3}$$

$$= \int_\Omega \langle C^s d(u), d(v)\rangle dx \;=\; a^s(\mathbf{u}, \mathbf{v}) \tag{4}$$

where

$$C = \begin{bmatrix} \lambda + 2\mu & 0 & 0 & \lambda \\ 0 & \mu & \mu & 0 \\ 0 & \mu & \mu & 0 \\ \lambda & 0 & 0 & \lambda + 2\mu \end{bmatrix}, \quad C^s = \begin{bmatrix} \lambda + 2\mu & 0 & 0 & \lambda + \mu \\ 0 & \mu & 0 & 0 \\ 0 & 0 & \mu & 0 \\ \lambda + \mu & 0 & 0 & \lambda + 2\mu \end{bmatrix}, \tag{5}$$

$$d(\mathbf{u}) \;=\; \left(\frac{\partial u_1}{\partial x_1}, \frac{\partial u_1}{\partial x_2}, \frac{\partial u_2}{\partial x_1}, \frac{\partial u_2}{\partial x_2} \right)^t . \tag{6}$$

Note that (4) holds due to homogeneous pure displacement boundary conditions as for $u, v \in \mathcal{V}$ we have $\int_\Omega \frac{\partial u_i}{\partial x_j}\frac{\partial v_j}{\partial x_i} dx \;=\; \int_\Omega \frac{\partial u_i}{\partial x_i}\frac{\partial v_j}{\partial x_j} dx$. The matrix C^s is positive definite. More details can be found e.g. in [3,10,11].

The variational formulation with the modified bilinear form a^s is next discretized using Crouzeix-Raviart non-conforming linear finite elements, i.e., the continuous space $\mathcal{V}$ is replaced by a finite dimensional subspace $\mathcal{V}^{(\ell)}$. For more details see [10,11]. Here, and in what follows, $\{\mathcal{V}^{(k)}\}_{k=0}^{\ell}$ and $\{A^{(k)}\}_{k=0}^{\ell}$ stand for the FE spaces and

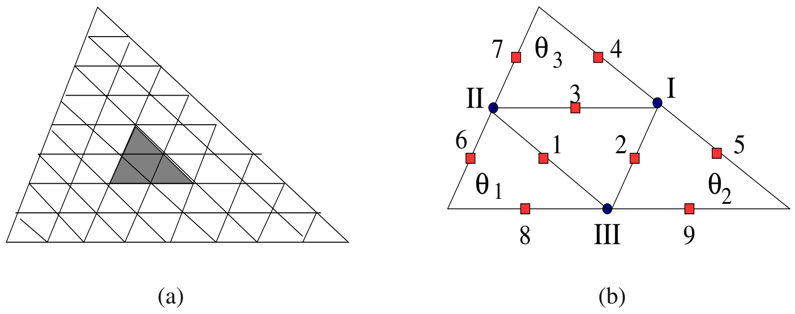


Fig. 1. Regular refinement (a), and macroelement local node numbering (b)

for the stiffness matrices corresponding to the triangulations $\{\mathcal{T}_k\}_{k=0}^{\ell}$. Let us recall that non-conforming FE approximations provide some attractive stability properties for parameter-dependent problems.

2 Hierarchical Decomposition of Crouzeix-Raviart Systems

Let us consider two consecutive mesh refinements $\mathcal{T}_k$ and $\mathcal{T}_{k+1}$. As already mentioned, for Crouzeix-Raviart non-conforming linear elements, the FE spaces associated with two consecutive mesh refinements are not nested. To enable the use of the general multilevel scheme, we consider the so called differences and aggregates (DA) approach to construct hierarchical two-level decomposition of the Crouzeix-Raviart systems. Such a technique was originally introduced and analysed in [8] for scalar elliptic problems. The algorithm is easily described on macroelement level, see Figure 1(b). Let $\phi_1, \ldots, \phi_9$ be the standard nodal non-conforming linear finite element basis functions on the macroelement E. Then for the 2D elasticity we use the basis functions

$$\phi_i^{(1)} = (\phi_i, 0)^t \text{ and } \phi_i^{(2)} = (0, \phi_i)^t, \quad i = 1, \ldots, 9.$$

The vector of the macroelement basis functions

$$\varphi_E = \{\Phi_i\}_{i=1}^{18} = \{\phi_1^{(1)}, \phi_1^{(2)}, \phi_2^{(1)}, \phi_2^{(2)}, \ldots, \phi_9^{(1)}, \phi_9^{(2)}\}$$

can be transformed into a vector of new hierarchical basis functions

$$\widetilde{\varphi}_E = \{\widetilde{\Phi}_i\}_{i=1}^{18}, \text{ where } \widetilde{\Phi}_i = \sum_j J_{ij}\Phi_j.$$

by using the transformation matrix $J_{DA} = (J_{ij})$,

$$J_{DA} = \left[\begin{array}{cccccccccccccccccccccccc}
1 \\
& 1 \\
& & 1 \\
& & & 1 \\
& & & & 1 \\
& & & & & 1 \\
& & & & & & 1 & 0 & & -1 \\
& & & & & & & 1 & & 0 & & -1 \\
& & & & & & & & & & & & 1 & 0 & & -1 \\
& & & & & & & & & & & & & 1 & & 0 & & -1 \\
& & & & & & & & & & & & & & & & & & 1 & 0 & & -1 \\
& & & & & & & & & & & & & & & & & & & 1 & & 0 & & -1 \\
1 & & & & & & 1 & 0 & & 1 \\
& 1 & & & & & & 1 & & 0 & & 1 \\
& & 1 & & & & & & & & & & 1 & 0 & & 1 \\
& & & 1 & & & & & & & & & & 1 & & 0 & & 1 \\
& & & & 1 & & & & & & & & & & & & & & 1 & 0 & & 1 \\
& & & & & 1 & & & & & & & & & & & & & & 1 & & 0 & & 1
\end{array}\right].$$

Now we are in a position to define the splitting $\mathcal{V}\,(E) = \{\Phi_i\}_{i=1}^{18} = \widetilde{\mathcal{V}}_1(E) \oplus \widetilde{\mathcal{V}}_2(E)\,,$

$$\begin{aligned}
\widetilde{\mathcal{V}}_1\,(E) &= \text{span}\,\{\,\widetilde{\Phi}_i\}_{i=1}^{12} \\
&= \text{span}\,\{\,\phi_1^{(k+1)},\,\phi_2^{(k+1)},\,\phi_3^{(k+1)}, \\
&\qquad \phi_4^{(k+1)} - \phi_5^{(k+1)},\,\phi_6^{(k+1)} - \phi_7^{(k+1)},\,\phi_8^{(k+1)} - \phi_9^{(k+1)}\}_{k=0}^{1},
\end{aligned} \tag{7}$$

$$\begin{aligned}
\widetilde{\mathcal{V}}_2\,(E) &= \text{span}\,\{\,\widetilde{\Phi}_i\}_{i=13}^{18} \\
&= \text{span}\,\{\,\phi_1^{(k+1)} + \phi_4^{(k+1)} + \phi_5^{(k+1)}, \\
&\qquad \phi_2^{(k+1)} + \phi_6^{(k+1)} + \phi_7^{(k+1)},\,\phi_3^{(k+1)} + \phi_8^{(k+1)} + \phi_9^{(k+1)}\}_{k=0}^{1}.
\end{aligned} \tag{8}$$

Accordingly, J_{DA} transforms the macroelement stiffness matrix A_E into a hierarchical form $\widetilde{A}_E = J\,A_E J^T$,

$$\widetilde{A}_E = \begin{bmatrix} \widetilde{A}_{E,11} & \widetilde{A}_{E,12} \\ \widetilde{A}_{E,21} & \widetilde{A}_{E,22} \end{bmatrix} \begin{array}{l} \widetilde{\phi}_i \in \widetilde{\mathcal{V}}_1(E) \\ \widetilde{\phi}_i \in \widetilde{\mathcal{V}}_2(E) \end{array}.$$

The corresponding global stiffness matrix

$$\widetilde{A}^{(k+1)} = \sum_{E \in \mathcal{T}_{k+1}} \widetilde{A}_E$$

can be again decomposed into 2×2 blocks

$$\widetilde{A}^{(k+1)} = \begin{bmatrix} \widetilde{A}_{11}^{(k+1)} & \widetilde{A}_{12}^{(k+1)} \\ \widetilde{A}_{21}^{(k+1)} & \widetilde{A}_{22}^{(k+1)} \end{bmatrix}, \tag{9}$$

which are induced by the decomposition on macroelement level. The block $\widetilde{A}_{11}^{(k+1)}$ corresponds to interior nodal unknowns with respect to the macro-elements $E \in \mathcal{T}_{k+1}$ plus differences of the nodal unknowns along the sides of $E \in \mathcal{T}_{k+1}$. The block $\widetilde{A}_{22}^{(k+1)}$ corresponds to certain aggregates of nodal unknowns. The related splitting of the current FE space reads as $\mathcal{V}^{(k+1)} = \widetilde{\mathcal{V}}_1^{(k+1)} \oplus \widetilde{\mathcal{V}}_2^{(k+1)}$. The introduced decomposition can be used to construct two-level preconditioners. The efficiency of these preconditioners depends (see, e.g. [12]) on the corresponding strengthened CBS constant γ_{DA}:

$$\gamma_{DA} = \sup \left\{ \frac{\left\langle \widetilde{A}_{12}^{(k+1)} y_2, y_1 \right\rangle}{\sqrt{\left\langle \widetilde{A}_{11}^{(k+1)} y_1, y_1 \right\rangle} \sqrt{\left\langle \widetilde{A}_{22}^{(k+1)} y_2, y_2 \right\rangle}},\ y_1 \neq 0,\ y_2 \neq 0 \right\}.$$

The following result holds.

Theorem 1. *Consider the problem where the Lamé coefficients are constant on the coarse triangles $E \in \mathcal{T}_k$, discretization by the Crouzeix-Raviart FE and the DA decomposition of the stiffness matrix. Then for any element size and shape and for any Poisson ratio $\nu \in (0, \frac{1}{2})$, there holds*

$$\gamma_{DA} \leq \sqrt{\frac{3}{4}}.$$

Proof. The constant γ_{DA} can be estimated by the maximum of the constants over the macroelements. Let us first consider the case of a right angled reference macroelement, see Figure 2.

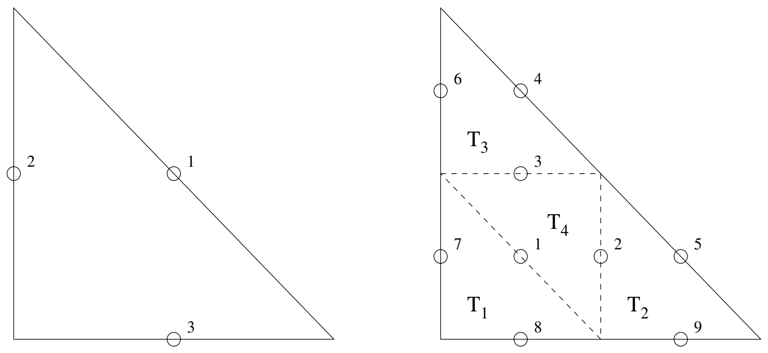


Fig. 2. The reference coarse grid triangle and the macroelement $\widehat{E}$

Let $\widetilde{\mathcal{V}}_1(\widehat{E})$, $\widetilde{\mathcal{V}}_2(\widehat{E})$, be the two-level splitting for the reference macroelement $\widehat{E}$. For $u \in \widetilde{\mathcal{V}}_1(\widehat{E})$ and $v \in \widetilde{\mathcal{V}}_2(\widehat{E})$ denote $d^{(r)} = d^{(r)}(u)\mid_{T_r}$ and $\delta^{(r)} = d^{(r)}(v)\mid_{T_r}$, $r = 1 \cdots, 4$, see the definition (6). Then it is easy to show (cf. [8]) that

$$d^{(1)} + d^{(2)} + d^{(3)} + d^{(4)} = 0, \tag{10}$$

$$\delta^{(1)} = \delta^{(2)} = \delta^{(3)} = -\delta^{(4)} = \delta. \tag{11}$$

Hence,

$$
\begin{aligned}
a_{\widehat{E}}(u,v) &= \sum_{r=1}^{4} \int_{T_r} \langle C^s d(u),\, d(v)\rangle\, dx \;=\; \sum_{r=1}^{4} \triangle \langle C^s d^{(r)},\, \delta^{(r)}\rangle \\
&= \triangle \langle C^s \delta,\, d^{(1)} + d^{(2)} + d^{(3)} - d^{(4)}\rangle \qquad (12)\\
&= -2\triangle \langle C^s\delta,\, d^{(4)}\rangle \le 2\triangle \mid \delta \mid_{C^s} \mid d^{(4)} \mid_{C^s}
\end{aligned}
$$

where $\triangle = \text{area}(T_k)$, $\langle x, y\rangle = x^T y$ denotes the inner product in R^4, and $\mid d \mid_{C^s} = \sqrt{\langle C^s d, d\rangle}$ is the norm induced by the coefficient matrix C^s. Further,

$$
\mid d^{(4)} \mid_{C^s}^2 = \mid d^{(1)} + d^{(2)} + d^{(3)} \mid_{C^s}^2 \le 3 \sum_{k=1}^{3} \mid d^{(k)} \mid_{C^s}^2
$$

leads to

$$
a_{\widehat{E}}(u,u) \;=\; \sum_{k=1}^{4} \mid d^{(k)} \mid_{C^s}^2 \triangle \;\ge\; \left(1 + \frac{1}{3}\right) \triangle \mid d^{(4)} \mid_{C^s}^2 \qquad (13)
$$

and

$$
a_{\widehat{E}}(v,v) \;=\; 4\triangle \mid \delta \mid_{C^s}^2 \;. \qquad (14)
$$

Thus,

$$
a_{\widehat{E}}(u,v) \le \sqrt{\frac{3}{4}} \sqrt{a_{\widehat{E}}(u,u)} \sqrt{a_{\widehat{E}}(v,v)}\,. \qquad (15)
$$

In the case of an arbitrary shaped macroelement E we can use the affine mapping $F : \widehat{E} \to E$ for transformation of the problem to the reference macroelement, for more details see e.g. [3]. This transformation changes the coefficient matrix C^s, but the estimate $\gamma_E \le \sqrt{\frac{3}{4}}$ still holds since the result (15) for the reference macroelement does not depend on the coefficient matrix C^s.

Remark 1. The obtained new uniform estimate of the CBS constant γ is a generalization of the earlier estimate from [14], namely

$$
\gamma \le \frac{\sqrt{8 + \sqrt{8}}}{4} \approx 0.822,
$$

which is derived in the case of a regular triangulation $\mathcal{T}_0$ obtained by a diagonal subdivision of a square mesh.

3 Multilevel Preconditioning

The standard multiplicative two-level preconditioner, based on (9), can be written in the form

$$
\widetilde{C}^{(k+1)} = \begin{bmatrix} \widetilde{A}_{11}^{(k+1)} & 0 \\ \widetilde{A}_{21}^{(k+1)} & \widetilde{A}_{22}^{(k+1)} \end{bmatrix} \begin{bmatrix} I_1 & (\widetilde{A}_{11}^{(k+1)})^{-1} \widetilde{A}_{12}^{(k+1)} \\ 0 & I_2 \end{bmatrix}. \qquad (16)
$$

The convergence rate of the related two-level iterative method is based on the spectral condition number estimate

$$\kappa((\widetilde{C}^{(k+1)})^{-1}\widetilde{A}^{(k+1)}) < \frac{1}{1-\gamma_{DA}^2} < 4.$$

The following theorem is useful for extending the two-level to multilevel preconditioners.

Theorem 2. *Consider again the elasticity problem with constant Lamé coefficients on the triangles $E \in \mathcal{T}_k$, discretization by the Crouzeix-Raviart FE and the DA decomposition of the stiffness matrix. Let $\widetilde{A}_{22}^{(k+1)}$ be the stiffness matrix corresponding to the space $\widetilde{\mathcal{V}}_2^{(k+1)}$ from the introduced DA splitting, and let $A^{(k)}$ be the stiffness matrix, corresponding to the coarser triangulation $\mathcal{T}_k$, equipped with the standard nodal finite element basis. Then*

$$\widetilde{A}_{22}^{(k+1)} = 4\,A^{(k)}\,. \tag{17}$$

The proof follows almost directly from the definitions of the hierarchical basis functions $\widetilde{\Phi}_i$ with value equal to one in two nodes of one of the macroelement sides and one opposite inner node and the corresponding coarse grid basis function with value equal to one in one node on the same side. This result enables the recursive multilevel extension of the considered two-level multiplicative preconditioner preserving the same estimate of the CBS constant. In particular, the general scheme of the algebraic multilevel iteration (AMLI) algorithm becomes straightforwardly applicable (see [6,7]).

We consider now the construction of optimal preconditioners for the coarse grid complement systems $\widetilde{C}_{11}^{(k+1)}$ for the blocks $\widetilde{A}_{11}^{(k+1)}$, see decomposition (9). We search for optimal preconditioners in the sense that they are spectrally equivalent to the top-left matrix block independently on mesh size, element shape and Poisson ratio. Moreover the cost of applying the preconditioner is aimed to be proportional to the number of degrees of freedom. Similarly to [4,5,9], we construct preconditioners on macroelement level and assemble the local contributions to obtain $\widetilde{C}_{11}^{(k+1)}$.

Remark: One possible approach is first to impose a displacement decomposition ordering, then use a block-diagonal approximation of $\widetilde{A}_{11}^{(k+1)}$, and then precondition the diagonal blocks which are elliptic. Let us assume that the multiplicative preconditioner from [9] is applied to the diagonal blocks of $\widetilde{A}_{11}^{(k+1)}$. Then, for homogeneous isotropic materials, the following simplified estimate holds

$$\kappa\left(\widetilde{B}_{11}^{(k+1)^{-1}}\,\widetilde{A}_{11}^{(k+1)}\right) \le \frac{1-\nu}{1-2\nu}\,\frac{15}{8}.$$

The presented construction is optimal with respect to mesh size and mesh anisotropy but is applicable for moderate values of $\nu \in (0, \frac{1}{2})$ only. When the material is *almost incompressible*, it is better to apply a macroelement level static condensation of $\widetilde{A}_{11}^{(k+1)}$ first, which is equivalent to the elimination of all unknowns corresponding to the interior nodes of the macroelements, see Figure 1(b). Let us assume that the triangulations $\mathcal{T}_0$ is constructed by diagonal subdivision of a square mesh. Let the corresponding Schur compliment be approximated by its diagonal. Then the resulted preconditioner satisfies the following estimate

$$\kappa\left(\widetilde{B}_{11}^{(k+1)^{-1}}\widetilde{A}_{11}^{(k+1)}\right) \leq const \approx 8.301\cdots ,$$

which is uniform with respect to the Poisson ratio ν (see for more details in [14]). The robustness of the later approach is demonstrated by the numerical tests presented in the next section.

4 Numerical Tests

It is well known, that if low order conforming finite elements are used in the construction of the approximation space, when the Poisson ratio ν tends to 0.5, the so called *locking phenomenon* appears. Following [10,11], we use the Crouzeix-Raviart linear finite elements to get a *locking-free* FEM solution of the problem. Note that the straightforward FEM discretization works well for the pure displacement problem only.

The presented numerical tests illustrate the behavior of the FEM error as well as the optimal convergence rate of the AMLI algorithm when the size of the discrete problem is varied and $\nu \in (0, 1/2)$ tends to the *incompressible* limit. We consider the simplest test problem in the unit square $\Omega = (0,1)^2$ with $E = 1$. The right hand side corresponds to the following exact solution $\mathbf{u}(x,y) = [\sin(\pi x)\sin(\pi y), y(y-1)x(x-1)]$. The relative stopping criterion for the PCG iterations is

$$(C_{AMLI}^{-1} r^{N_{it}}, r^{N_{it}})/(C_{AMLI}^{-1} r^0, r^0) < \varepsilon^2,$$

where r^i stands for the residual at the i-th iteration step.

The relative FEM errors, given in Table 1, illustrates very well the *locking-free* approximation. Here the number of refinement steps is $\ell = 4$, $N = 1472$, and $\varepsilon = 10^{-9}$. In Table 2, the number of iterations are presented as a measure of the robustness of the multilevel preconditioner. The optimal order *locking-free* convergence rate of the AMLI algorithm is well expressed. Here β stands for the degree of the stabilization polynomial. The value of $\beta = 2$ corresponds to the derived uniform estimate of the CBS constant, providing the total computational cost optimality of the related PCG algorithm.

Table 1. Relative error stability for $\nu \to 1/2$

ν	$\|u-u_h\|_{[L_2]^2}/\|f\|_{[L_2]^2}$	ν	$\|u-u_h\|_{[L_2]^2}/\|f\|_{[L_2]^2}$
0.4	.3108249106503572	0.4999	.3771889077038727
0.49	.3695943747405575	0.49999	.3772591195613628
0.499	.3764879643773666	0.499999	.3772661419401481

Table 2. Iterations: $\varepsilon = 10^{-3}$, $\beta = 2$

ℓ	$N \diagdown \nu$	0.3	0.4	0.49	0.499	0.4999	0.49999	0.499999
4	1472	13	13	12	13	13	13	13
5	6016	12	12	12	14	13	13	13
6	24320	12	12	12	12	13	13	13
7	97792	11	11	11	12	13	13	13
8	196096	11	11	11	12	12	13	13

5 Concluding Remarks

This study is strongly motivated by the expanding interest in non-conforming finite elements, which are very useful for solving problems, where the standard conforming elements may suffer from the so-called *locking effects*. The success of the Crouzeix-Raviart and other non-conforming finite elements can be explained also by the fact that they produce algebraic systems that are equivalent to the Schur complement system for the Lagrange multipliers arising from the mixed finite element method for Raviart-Thomas elements. There are also other advantages of the non-conforming Crouzeix-Raviart finite elements, such as less density of the stiffness matrix etc. The developed robust PCG algorithms are further applicable as efficient preconditioning operators in the context of nonlinear elliptic problems, see [13].

The presented multilevel algorithm has some well expressed inherently parallel features. The key point here is that the considered approximations of the coarse grid complement blocks $\widetilde{A}_{11}$ are of either diagonal or generalized tridiagonal form. The observed potential for parallel implementation is of a further importance for efficient solution of large-scale real-life problems including coupled long-time models which are as a rule nonlinear.

Acknowledgments

This work is supported in part by the Centre of Excellence BIS-21 grant ICA1-2000-70016 and the Bulgarian NSF grant IO-01/2003.

References

1. B. Achchab, O. Axelsson, L. Laayouni, A. Souissi. Strengthened Cauchy-Bunyakowski-Schwarz inequality for a three dimensional elasticity system, *Numerical Linear Algebra with Applications*, Vol. 8(3): 191-205, 2001.
2. B. Achchab, J.F. Maitre. Estimate of the constant in two strengthened CBS inequalities for FEM systems of 2D elasticity: Application to Multilevel methods and a posteriori error estimators, *Numerical Linear Algebra with Applications*, Vol. 3 (2): 147-160, 1996.
3. O. Axelsson, R. Blaheta. Two simple derivations of universal bounds for the C.B.S. inequality constant, *Applications of Mathematics*, Vol.49 (1): 57-72, 2004.

4. O. Axelsson, S. Margenov. On multilevel preconditioners which are optimal with respect to both problem and discretization parameters, *Computational Methods in Applied Mathematics*, Vol. 3, No. 1: 6-22, 2003.
5. O. Axelsson, A. Padiy. On the additive version of the algebraic multilevel iteration method for anisotropic elliptic problems, *SIAM Journal on Scientific Computing* 20, No.5: 1807-1830, 1999.
6. O. Axelsson and P.S. Vassilevski. Algebraic Multilevel Preconditioning Methods I, *Numerische Mathematik*, 56: 157-177, 1989.
7. O. Axelsson and P.S. Vassilevski. Algebraic Multilevel Preconditioning Methods II, *SIAM Journal on Numerical Analysis*, 27: 1569-1590, 1990.
8. R. Blaheta, S. Margenov, M. Neytcheva. Uniform estimate of the constant in the strengthened CBS inequality for anisotropic non-conforming FEM systems, *Numerical Linear Algebra with Applications*, 11: 309-326,2004.
9. R. Blaheta, S. Margenov, M. Neytcheva. Robust optimal multilevel preconditioners for non-conforming finite element systems, to appear in *Numerical Linear Algebra with Applications*.
10. S. Brenner and L. Scott. The mathematical theory of finite element methods, *Texts in applied mathematics, vol. 15, Springer-Verlag*, 1994.
11. S. Brenner and L. Sung. Linear finite element methods for planar linear elasticity, *Math. Comp.*, 59: 321-338, 1992.
12. V. Eijkhout and P.S. Vassilevski. The role of the strengthened Cauchy-Bunyakowski-Schwarz inequality in multilevel methods, *SIAM Review*, 33: 405-419, 1991.
13. I. Farago, J. Karatson. Numerical solution of nonlinear elliptic problems via preconditionionig operators. Theory and applications, *NOVA Science*, 2002.
14. Tz. Kolev, S. Margenov. Two-level preconditioning of pure displacement non-conforming FEM systems, *Numerical Linear Algebra with Applications*, 6: 533-555, 1999.
15. Tz. Kolev, S. Margenov. AMLI preconditioning of pure displacement non-conforming elasticity FEM systems, *Springer LNCS*, 1988: 482-489, 2001.
16. S. Margenov. Upper bound of the constant in the strengthened C.B.S. inequality for FEM 2D elasticity equations, *Numer. Linear Algebra Appl.*, 1: 65-74, 1994.
17. S. Margenov, P.S. Vassilevski. Algebraic multilevel preconditioning of anisotropic elliptic problems, *SIAM Journal on Scientific Computing*, V.15(5): 1026-1037, 1994.

Efficient Solvers for 3-D Homogenized Elasticity Model

Ronald H.W. Hoppe and Svetozara I. Petrova

Institute of Mathematics, University of Augsburg
D-86159 Augsburg, Germany
{hoppe,petrova}@math.uni-augsburg.de

Abstract. The optimization of the macroscopic behavior of microstructured materials using microscopic quantities as design variables is a well established discipline in materials science. The paper deals with recently produced microcellular biomorphic ceramics. The mechanical macromodel corresponding to these composite materials is obtained by homogenization. The homogenized elasticity tensor and its dependence on the design variables are computed numerically involving adaptive finite element approximations of elasticity problems in the 3-D periodicity cell. Efficient iterative solvers based on incomplete Cholesky (IC) decomposition and algebraic multigrid method (AMG) as preconditioners of the stiffness matrix are proposed in the application of PCG method.

1 Introduction

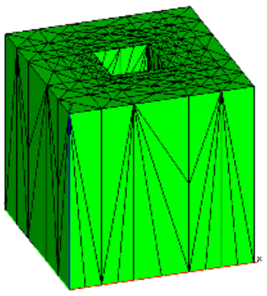
The production of microcellular biomorphic ceramics by biotemplating processes is a particular area within biomimetics which has emerged as a perspective new technology in materials science during the past decade (cf., e.g., [11]). The biological object under consideration in this paper is naturally grown wood which is known to be highly porous and to possess excellent mechanical properties. The wood morphologies are characterized by an open porous system of tracheidal cells which provide the transportation path for water and minerals in the living plants. The biotemplating process uses wooden specimen to produce graphite-like carbon preforms by high temperature pyrolysis followed by an infiltration by liquid-phase or gaseous-phase materials such as silicon (Si) or titanium (Ti) to come up with SiC- or TiC-ceramics (see, e.g., [6] for details). An important feature of the biotemplating process is that it preserves the high porosity of the wooden specimen and results in a final ceramics with excellent structural-mechanical and thermomechanical properties which can be used as heat insulators, particle filters, catalyst carriers, automotive tools, and medical implants.

The macroscopic mechanical behavior of the microcellular biomorphic ceramics depends on microscopic geometrical quantities such as the size of the voids and the lengths and widths of the different layers forming the cell walls. While the size of the voids is determined by the growth of the wood itself (early/late wood), the other quantities can be influenced by tuning the parameters of the biotemplating process. Therefore, an optimal structural design of the ceramics can be performed where the state equation is given by linear elasticity and the design variables are chosen as the microstructural geometrical quantities (cf., e.g., [8]). The objective functional depends on the mode of loading. Since the resolution of the microstructures is cost prohibitive with respect to computational work, the idea is to derive a homogenized macromodel featuring the dependence on the microstructural design variables and to apply the optimization process to the homogenized model.

2 Computation of Homogenized Elasticity Tensor

For the structural optimization of the microcellular biomorphic SiC ceramics modern optimization techniques (see, [7]) are applied to the mechanical macromodel obtained by the homogenization approach (cf., e.g., [3,9]).

We assume the workpiece of macroscopic length L to consist of periodically distributed constituents with a cubic periodicity cell Y of microscopic characteristic length ℓ consisting of an interior void channel (V) surrounded by layers of silicon carbide (SiC) and carbon (C) (cf. Fig.1).



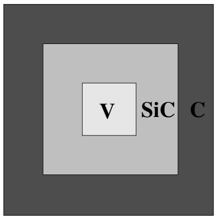


Fig. 1. a) Periodicity cell $Y = [0,\ell]^3$, b) Cross section of $Y = V \cup SiC \cup C$

Assuming linear elasticity and denoting by $\mathbf{u}$ the displacements vector, the stress tensor $\boldsymbol{\sigma}$ is related to the linearized strain tensor $\mathbf{e} = \frac{1}{2}(\nabla\mathbf{u} + (\nabla\mathbf{u})^T)$ by Hooke's law $\boldsymbol{\sigma} = \mathbf{E}\mathbf{e}$, where $\mathbf{E} = \mathbf{E}(X) = (E_{ijk\ell}(X))$ stands for the elasticity tensor whose components attain different values in the regions V, SiC, and C:

$$\mathbf{E} = \begin{pmatrix} E_{1111} & E_{1122} & E_{1133} & E_{1112} & E_{1123} & E_{1113} \\ & E_{2222} & E_{2233} & E_{2212} & E_{2223} & E_{2213} \\ & & E_{3333} & E_{3312} & E_{3323} & E_{3313} \\ & & & E_{1212} & E_{1223} & E_{1213} \\ & & & & E_{2323} & E_{2313} \\ \text{SYM} & & & & & E_{1313} \end{pmatrix} \tag{2.1}$$

Introducing $x := X/L$ and $y := X/\ell$ as the macroscopic and microscopic variables and $\varepsilon := \ell/L$ as the scale parameter, homogenization based on the standard double scale asymptotic expansion results in the homogenized elasticity tensor $\mathbf{E}^H = (E^H_{ijk\ell})$ whose components are given by

$$E^H_{ijk\ell} = \frac{1}{|Y|} \int\limits_Y \left(E_{ijkl}(y) - E_{ijpq}(y) \frac{\partial \xi_p^{k\ell}}{\partial y_q}\right) dy \; . \tag{2.2}$$

The tensor $\boldsymbol{\xi}=(\xi_p^{k\ell}),\ k,l,p = 1,2,3$, with periodic components $\xi_p^{k\ell} \in H^1_{per}(Y)$ has to be computed via the solution of the elasticity problems

$$\int_Y \left(E_{ijpq}(y) \frac{\partial \xi_p^{kl}}{\partial y_q} \right) \frac{\partial \phi_i}{\partial y_j} \, dy = \int_Y E_{ijkl}(y) \frac{\partial \phi_i}{\partial y_j} \, dy \tag{2.3}$$

for an arbitrary $Y-$periodic variational function $\boldsymbol{\phi} \in \mathbf{H}^1(Y)$. We note that explicit formulas for the homogenized elasticity tensor are only available in case of laminated or checkerboard structures (cf., e.g., [2,9]). Therefore, (2.3) has to be solved numerically which has been done by using continuous, piecewise linear finite elements with respect to adaptively generated locally quasi-uniform and shape regular simplicial tetrahedral partitionings of the periodicity cell Y.

3 Mesh Adaptivity by a Posteriori Error Estimation

The computation of the homogenized elasticity coefficients requires the solution of linear elastic boundary value problems with the periodicity cell Y as the computational domain. Due to the composite character of the cell there are material interfaces where the solution changes significantly. Hence, local refinement of the underlying finite element mesh is strongly advised. In contrast to previous work in structural optimization (cf., e.g., [1,2]) where local refinement is done by manual remeshing, we have used an automatic grid refinement based on a posteriori error estimator of Zienkiewicz-Zhu type [13] obtained by local averaging of the computed stress tensor.

Using an approximation of the components of the displacements vector by continuous, piecewise linear finite elements with respect to a simplicial tetrahedrization $\mathcal{T}_h$ of the periodicity cell Y, we denote by $\hat{\sigma}$ the discontinuous finite element stress. A continuous recovered stress σ^* is obtained at each nodal point p by local averaging: Denoting by $Y_p \subset Y$ the union of all elements $K \in \mathcal{T}_h$ sharing p as a vertex, we compute

$$\sigma^*(p) = \sum_{K \in Y_p} \omega_K \, \hat{\sigma}|_K \quad , \quad \omega_K := \frac{|K|}{|Y_p|} \; , \; K \in Y_p \; . \tag{3.4}$$

Based on (3.4), we have chosen

$$\eta := \left(\sum_{K \in \mathcal{T}_h} \eta_K^2 \right)^{1/2} \quad , \quad \eta_K := \|\sigma^* - \hat{\sigma}\|_{0,K} \; , \; K \in \mathcal{T}_h \tag{3.5}$$

as a global estimator whose local contributions η_K are cheaply computable.

Note that such an estimator has been studied and analyzed in [10] for linear second order elliptic boundary value problems where it was shown that η is asymptotically exact. Moreover, general averaging techniques for low order finite element approximations of linear elasticity problems have been considered in [5] and efficiency and reliability of Zienkiewicz-Zhu type estimators have been established.

4 Iterative Solution Techniques

After finite element discretization of the domain Y the elasticity equation (2.3) used to compute the effective coefficients results in the following system of linear equations

$$A\,\mathbf{u} = \mathbf{f}, \tag{4.6}$$

where $\mathbf{u}$ is the vector of unknown displacements and $\mathbf{f}$ is the discrete right–hand side. The stiffness matrix A is symmetric and positive definite but not an M-matrix. Two typical orderings of the unknowns are often used in practice, namely

$$\left(u_1^{(x)}, u_1^{(y)}, u_1^{(z)}, u_2^{(x)}, u_2^{(y)}, u_2^{(z)}, \ldots, u_n^{(x)}, u_n^{(y)}, u_n^{(z)}\right), \tag{4.7}$$

referred to as a *pointwise displacements ordering* and

$$\left(u_1^{(x)}, u_2^{(x)}, \ldots, u_n^{(x)}, u_1^{(y)}, u_2^{(y)}, \ldots, u_n^{(y)}, u_1^{(z)}, u_2^{(z)}, \ldots, u_n^{(z)}\right), \tag{4.8}$$

called the *separate displacements ordering*. Here, $u_k^{(x)}$, $u_k^{(y)}$, and $u_k^{(z)}$ are the corresponding x, y-, and z- displacement components.

Using (4.8), for instance, the matrix A admits the following 3×3 block decomposition

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}. \tag{4.9}$$

In case of isotropic materials, the diagonal blocks $A_{jj},\ j = 1,2,3$, in (4.9) are discrete analogs of the following anisotropic Laplacian operators

$$\tilde{D}_1 = a\frac{\partial^2}{\partial x^2} + b\frac{\partial^2}{\partial y^2} + b\frac{\partial^2}{\partial z^2},\ \tilde{D}_2 = b\frac{\partial^2}{\partial x^2} + a\frac{\partial^2}{\partial y^2} + b\frac{\partial^2}{\partial z^2},\ \tilde{D}_3 = b\frac{\partial^2}{\partial x^2} + b\frac{\partial^2}{\partial y^2} + a\frac{\partial^2}{\partial z^2}$$

with coefficients $a = E(1-\nu)/((1+\nu)(1-2\nu))$ and $b = 0.5E/(1+\nu)$ where E is the Young modulus and ν is the Poisson ratio of the corresponding material. This anisotropy requires a special care to construct an efficient preconditioner for the iterative solution method. Based on Korn's inequality, it can be shown that A and its block diagonal part are spectrally equivalent. The condition number of the preconditioned system depends on the Poisson ratio ν of the materials and the constant in the Korn inequality. For the background of the spectral equivalence approach using block diagonal displacement decomposition preconditioners in linear elasticity problems we refer to [4]. Note that the spectral equivalence estimate will deteriorate for ν close to 0.5 which is not the case in our particular applications.

The PCG method is applied to solve the linear system (4.6). We propose two approaches to construct a preconditioner for A:

(i) construct a preconditioner for the global matrix A

(ii) construct a preconditioner for A of the type

$$M = \begin{bmatrix} M_{11} & 0 & 0 \\ 0 & M_{22} & 0 \\ 0 & 0 & M_{33} \end{bmatrix}, \tag{4.10}$$

where $M_{jj} \sim A_{jj},\ j = 1, 2, 3$, are "good" approximations to the diagonal blocks of A. In case (i) we have chosen the incomplete Cholesky (IC) factorization of A with an appropriate stopping criterion.

An efficient preconditioner for A_{jj} in case (ii) turns out to be a matrix M_{jj} corresponding to a Laplacian operator $(-\text{div}\,(c\,\text{grad}\,u))$ with a fixed scale factor c. In our case we use, for instance, $c = b/2$ for all three diagonal blocks. *Algebraic multigrid* (AMG) method is applied as a "plug-in" solver for A (see [12] for details). This method is a purely matrix–based version of the algebraic multilevel approach and has shown in the last decade numerous efficient implementations in solving large sparse unstructured linear systems of equations without any geometric background.

5 Numerical Experiments

In this section, we present some computational results concerning the microscopic problem to find the homogenized elasticity coefficients. The elasticity equation (2.3) is solved numerically using initial decomposition of the periodic microcell Y into hexahedra and then continuous, piecewise linear finite elements on tetrahedral shape regular meshes. Due to the equal solutions $\boldsymbol{\xi}^{12} = \boldsymbol{\xi}^{21},\ \boldsymbol{\xi}^{23} = \boldsymbol{\xi}^{32},$ and $\boldsymbol{\xi}^{13} = \boldsymbol{\xi}^{31}$ one has to solve six problems in the period Y to find $\boldsymbol{\xi}^{11}$ (Problem 1), $\boldsymbol{\xi}^{22}$ (Problem 2), $\boldsymbol{\xi}^{33}$ (Problem 3), $\boldsymbol{\xi}^{12}$ (Problem 4), $\boldsymbol{\xi}^{23}$ (Problem 5), and $\boldsymbol{\xi}^{13}$ (Problem 6). The discretized problems have been solved by the iterative solvers discussed in Section 4 and the mesh adaptivity has been realized by means of a Zienkiewicz-Zhu type a posteriori error estimator [13].

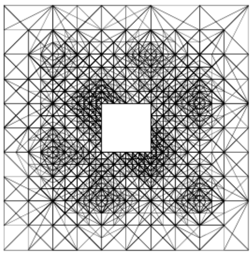


Fig. 2. Cross section of Y, density = 96%, Problem 3, $nt = 12395$, $nn = 2692$

The Young modulus E (in GPa) and the Poisson ratio ν of our two materials are, respectively, $E = 10,\ \nu = 0.22$ for carbon and $E = 410,\ \nu = 0.14$ for SiC. We denote by nt the number of tetrahedra and by nn the number of nodes on the corresponding refinement level. In Fig.2 the adaptive mesh refinement is visualized on the cross section of the period Y. Tables 1 and 2 contain information for the computed homogenized coefficients according to the refinement level.

Table 3 presents some convergence results for the proposed preconditioners within PCG method. For various values of the density μ of the periodicity microstructure we report the number of degrees of freedom d.o.f., the number of iterations iter, and the CPU–time in seconds for the first 11 adaptive refinement levels. One can see from the numerical results a better convergence of AMG–preconditioner compared to IC–factorization. We observe an essential efficiency of AMG for a larger number of unknowns.

Table 1. Homogenized coefficients w.r.t. adaptive refinement level, $\mu = 19\%$

level	E^H_{1111}	E^H_{2222}	E^H_{3333}	nt/nn (Prob.1)	nt/nn (Prob.2)	nt/nn (Prob.3)
1	160.82	174.34	204.39	288/126	288/126	288/126
2	175.60	207.65	214.79	334/137	332/136	332/136
3	159.18	170.78	206.73	443/166	457/169	441/165
4	174.07	166.79	213.77	637/222	593/208	595/211
5	168.97	163.26	214.10	982 /297	971/292	948/287
6	146.50	147.22	208.64	1641/433	1609/431	1684/443
7	160.25	147.90	211.11	2516/624	2422/604	2427/601
8	146.80	138.09	211.82	3761/896	3915/927	3881/920
9	137.10	134.55	210.36	5134/1171	7743/1722	5092/1160
10	133.22	131.84	210.91	10839/2259	13698/2869	11078/2289

Table 2. Homogenized coefficients for late wood, density $\mu = 91\%$

level	E^H_{1111}	E^H_{2222}	E^H_{3333}	E^H_{1212}	E^H_{2323}	E^H_{1313}
1	148.35	152.57	153.96	60.22	62.46	59.50
2	154.34	162.64	162.77	69.71	71.31	65.79
3	142.66	148.42	162.79	60.51	65.26	63.23
4	145.84	137.61	161.70	53.91	59.04	62.92
5	127.99	134.32	161.43	49.41	56.19	56.49
6	98.29	111.65	160.71	40.44	46.14	48.45
7	91.79	90.23	158.29	35.70	43.69	46.03
8	82.42	83.00	160.57	30.59	41.03	43.70
9	75.05	75.11	160.22	26.93	39.75	40.97
10	69.66	70.30	159.82	25.47	37.16	39.30

Table 3. Convergence results with IC and AMG preconditioners, density μ, Problem 1

prec.	level	1	2	3	4	5	6	7	8	9	10	11
$\mu = 51\%$	d.o.f.	78	90	126	225	336	579	1185	1908	3360	5598	9987
IC	iter	9	8	14	23	40	66	105	150	235	269	299
	CPU	e-16	e-16	e-16	0.1	0.2	0.2	0.9	2.4	8.2	20.9	59.1
AMG	iter	11	13	13	15	18	23	38	57	89	94	99
	CPU	e-16	e-16	e-16	0.2	0.3	0.5	1.5	3	7.6	14.8	23.5
$\mu = 84\%$	d.o.f.	78	93	150	261	510	1047	2103	3843	6537	10485	18459
IC	iter	10	11	16	21	44	78	117	171	226	273	301
	CPU	e-16	e-16	0.1	0.1	0.1	0.6	2.4	8.4	24.3	63.7	187.1
AMG	iter	12	14	14	14	18	31	43	73	69	74	75
	CPU	e-16	e-16	e-16	0.2	0.4	1.1	3	7.5	15.5	25.6	33.8

Acknowledgments

This work has been partially supported by the German National Science Foundation (DFG) under Grant No.HO877/5-3. The second author has also been supported in part by the Bulgarian NSF under Grant I1402/2004.

References

1. M.P. Bendsøe and N. Kikuchi. Generating optimal topologies in structural design using a homogenization method. *Comput. Methods Appl. Mech. Eng.*, 71:197–224, 1988.
2. M.P. Bendsøe and O. Sigmund. Topology Optimization: Theory, Methods and Applications. Springer, Berlin-Heidelberg-New York, 2003.
3. A. Bensoussan, J. L. Lions, G. Papanicolaou. Asymptotic Analysis for Periodic Structures. North-Holland, Elsevier Science Publishers, Amsterdam, 1978.
4. R. Blaheta. Displacement decomposition – incomplete factorization preconditioning techniques for linear elasticity problems. *Numer. Linear Algebra Appl.*, 1(2):107–128, 1994.
5. C. Carstensen and St. Funken. Averaging technique for FE-a posteriori error control in elasticity. Part II: λ-independent estimates. *Comput. Methods Appl. Mech. Engrg.*, 190:4663–4675, 2001.
6. P. Greil, T. Lifka, A. Kaindl. Biomorphic cellular silicon carbide ceramics from wood: I. Processing and microstructure, and II. Mechanical properties. *J. Europ. Cer. Soc.*. 18:1961–1973 and 1975–1983, 1998.
7. R.H.W. Hoppe and S.I. Petrova. Applications of primal-dual interior methods in structural optimization. *Comput. Methods Appl. Math.*, 3(1):159–176, 2003.
8. R.H.W. Hoppe and S.I. Petrova. Optimal shape design in biomimetics based on homogenization and adaptivity. *Math. Comput. Simul.*, 65(3):257–272, 2004.
9. V.V. Jikov, S.M. Kozlov, and O.A. Oleinik. Homogenization of Differential Operators and Integral Functionals. Springer, 1994.
10. R. Rodriguez. Some remarks on the Zienkiewicz-Zhu estimator. *Numer. Meth. PDEs*, 10:625–635, 1994.
11. M. Sarikaya and I.A. Aksay. Biomimetics: Design and Processing of Materials. AIP Series in Polymer and Complex Materials, Woodbury (New York), 1995
12. K. Stüben. A review of algebraic multigrid. *J. Comput. Appl. Math.*, 128:281–309, 2001.
13. O.C. Zienkiewicz and J.Z. Zhu. A simple error estimator and adaptive procedure for practical engineering analysis. *Intern. J. Numer. Methods Eng.*, 24:337–357, 1987.

Performance Evaluation of a Parallel Algorithm for a Radiative Transfer Problem*

Paulo B. Vasconcelos and Filomena d'Almeida

[1] Faculdade de Economia da Universidade do Porto
rua Dr. Roberto Frias s/n, 4200-464 Porto, Portugal
pjv@fep.up.pt

[2] Faculdade de Engenharia da Universidade do Porto
rua Dr. Roberto Frias s/n, 4200-465 Porto, Portugal
falmeida@fe.up.pt

Abstract. The numerical approximation and parallelization of an algorithm for the solution of a radiative transfer equation modeling the emission of photons in stellar atmospheres will be described. This is formulated in the integral form yielding a weakly singular Fredholm integral equation defined on a Banach space. The main objective of this work is to report on the performance of the parallel code.

Keywords: High performance computing, Fredholm integral equation, weakly singular kernel, projection approximation, iterative refinement, numerical methods.

AMS Subject Classification: 32A55, 45B05, 65D20, 65R20, 68W10.

1 The Integral Problem

We will consider the integral problem $T\varphi = z\varphi + f$ defined on a Banach space $X = L^1([0, \tau^*])$, where T is the integral operator

$$(Tx)(\tau) = \int_0^{\tau^*} g(|\tau - \tau'|)x(\tau')d\tau', \quad 0 \le \tau \le \tau^*, \tag{1.1}$$

the kernel g being a weakly singular decreasing function on $]0, \tau^*]$ and z belonging to the resolvent set of T.

This integral equation represents a radiative transfer problem. The emission of photons in stellar atmospheres is modeled by a strongly coupled system of nonlinear equations and equation (1.1) is a restriction of it by considering the temperature and pressure given (see [2] and [11] for details). For this astrophysics problem the free term f, is taken to be $f(\tau) = -1$ if $0 \le \tau \le \tau^*/2$, $f(\tau) = 0$ if $\tau^*/2 < \tau \le \tau^*$ and the kernel g is defined by

$$g(\tau) = \frac{\varpi}{2} E_1(\tau), \tag{1.2}$$

* This work was supported by CMUP. CMUP is financed by FCT under programs POCTI and POSI from QCA III with FEDER and National funds.

where

$$E_1(\tau) = \int_1^\infty \frac{\exp(-\tau\mu)}{\mu} d\mu, \tau > 0. \tag{1.3}$$

This is the first of the sequence of functions

$$E_\nu(\tau) = \int_1^\infty \frac{\exp(-\tau\mu)}{\mu^\nu} d\mu, \tau \geq 0, \nu > 1, \tag{1.4}$$

which have the following property: $E'_{\nu+1} = -E_\nu$ and $E_\nu(0) = \dfrac{1}{\nu - 1}, \nu > 1$ [1]. E_1 has a logarithmic singularity at $\tau = 0$.

The numerical solution is based on the projection of the integral operator onto a finite dimensional subspace. To obtain a good accuracy it is necessary to use a large dimension and in consequence a large algebraic linear system. The approach we use is to refine iteratively an approximate initial solution obtained by the projection onto a subspace of moderate (small) size. This iterative refinement is a Newton-type method where the resolvent operator is replaced with three different approximations. In opposition to the usual integral problems, this one leads to a band sparse discretization matrix. This fact is taken into account by using sparse data handling and by a particular distribution of the matrix blocks among the processors.

2 Projection and Iterative Refinement

Integral equations of this type are usually solved by discretization, for instance by projection methods, onto a finite dimensional subspace. The operator T is thus approximated by T_n, its projection onto the finite dimensional subspace X_n spanned by n linearly independent functions in X. In this case we will consider in X_n the basis of piecewise constant functions on each subinterval of $[0, \tau^*]$ determined by a grid of $n + 1$ points $0 = \tau_{n,0} < \tau_{n,1} < \ldots < \tau_{n,n} = \tau^*$. We get

$$T_n(x) = \sum_{j=1}^{n} \langle Tx, e^*_{n,j} \rangle e_{n,j}, \tag{2.5}$$

for all $x \in X$, where $e^*_{n,j} \in X^*$ (the adjoint space of X) and

$$\langle x, \ e^*_{n,j} \rangle = \frac{1}{h_{n,j}} \int_{\tau_{n,j-1}}^{\tau_{n,j}} x(\tau) d\tau, \tag{2.6}$$

$h_{n,j} = \tau_{n,j} - \tau_{n,j-1}$.

The approximate problem

$$(T_n - zI)\varphi_n = f \tag{2.7}$$

is then solved by means of an algebraic linear system of equations

$$(A_n - zI)x_n = b_n, \tag{2.8}$$

where A_n is a non singular matrix of dimension n. The relation between x_n and φ_n is given by $\varphi_n = \dfrac{1}{z}\left(\displaystyle\sum_{j=1}^{n} x_n(j)e_{n,j} - f\right)$.

In order to obtain an approximate solution φ_n with very good accuracy by this method it may be necessary to use a very large dimensional linear system.

Alternatively we may compute an initial approximation using a linear system of small dimension corresponding to a coarse grid on $[0,\tau^*]$ and refine it by a refinement formula based on a Newton-type method applied to the problem formulated under the form $Tx - zx - f = 0$. The Fréchet derivative of the left hand side operator is approximated by an operator F_n built on the coarse grid set on the domain: $x^{(k+1)} = x^{(k)} - F_n^{-1}(Tx^{(k)} - zx^{(k)} - f)$. The first choice for F_n is $(T_n - zI)$, F_n^{-1} being $R_n(z)$ the resolvent operator of T_n which approximates $R(z)$ if n is large enough. F_n^{-1} can also be replaced with $\dfrac{1}{z}(R_n(z)T - I)$ or $\dfrac{1}{z}(TR_n(z) - I)$.

After some algebraic manipulations we can set the corresponding three refinement schemes under the form [5,7,9]:

$$\text{Scheme 1:} \begin{cases} x^{(0)} = R_n(z)f \\ x^{(k+1)} = x^{(0)} + R_n(z)(T_n - T)x^{(k)}, \end{cases}$$

$$\text{Scheme 2:} \begin{cases} x^{(0)} = \dfrac{1}{z}(R_n(z)T - I)f \\ x^{(k+1)} = x^{(0)} + \dfrac{1}{z}R_n(z)(T_n - T)Tx^{(k)}, \end{cases}$$

$$\text{Scheme 3:} \begin{cases} x^{(0)} = \dfrac{1}{z}T(R_n(z) - I)f \\ x^{(k+1)} = x^{(0)} + \dfrac{1}{z}TR_n(z)(T_n - T)x^{(k)}. \end{cases}$$

A finer grid of $m+1$ points $0 = \tau_{m,0} < \tau_{m,1} < \ldots < \tau_{m,m} = \tau^*$ is set to obtain a projection operator T_m which is only used to replace the operator T in the schemes above and not to solve the corresponding approximate equation (2.7) with dimension m. The accuracy of the refined solution by the three refinement formulae is the same that would be obtained by applying the projection method directly to this fine grid operator (see [3]).

The convergence theorems and error bounds can be found in [4].

3 Matrix Computations

The evaluation of T cannot be done in closed form so T will be replaced with operator T_m corresponding to the projection of T onto X_m with $m \gg n$. The functional basis in X_n is $\{e_{n,j}\}_{j=1,\ldots,n}$ and in X_m it is $\{e_{m,j}\}_{j=1,\ldots,m}$, and the matrices representing the operators T_n and T_m restricted to X_n and X_m are respectively A_n and A_m. By observing the schemes 1, 2 and 3 we see that the restrictions of T_n to X_m and of T_m to X_n are also needed, they will be denoted by C and D respectively.

To simplify the description of the elements of the matrices we will assume that the coarse grid is a subset of the fine grid and we denote by q the ratio m/n. For matrix D we have

$$\begin{aligned} D(i,j) &= \frac{\varpi}{2h_{m,i}} \int_{\tau_{m,i-1}}^{\tau_{m,i}} \int_0^{\tau^*} E_1\left(|\tau-\tau'|\right) e_{n,j}\left(\tau'\right) d\tau' d\tau \\ &= \frac{\varpi}{2h_{m,i}} \int_{\tau_{m,i-1}}^{\tau_{m,i}} \int_{\tau_{n,j-1}}^{\tau_{n,j}} E_1\left(|\tau-\tau'|\right) d\tau' d\tau \end{aligned} \tag{3.9}$$

for $i = 1, ..., m$ and $j = 1, ..., n$.

$$\begin{aligned} D(i,j) = & \\ &= \frac{\varpi}{2h_{m,i}} \int_{\tau_{m,i-1}}^{\tau_{m,i}} (-E_2\left(\tau_{n,j}-\tau\right) + E_2\left(\tau_{n,j-1}-\tau\right)) d\tau \\ &\qquad \text{if } \tau_{m,i-1} \le \tau_{m,i} \le \tau_{n,j-1} \le \tau_{n,j} \\ &= \frac{\varpi}{2h_{m,i}} \int_{\tau_{m,i-1}}^{\tau_{m,i}} (E_2\left(\tau-\tau_{n,j}\right) - E_2\left(\tau-\tau_{n,j-1}\right)) d\tau \\ &\qquad \text{if } \tau_{n,j-1} \le \tau_{n,j} \le \tau_{m,i-1} \le \tau_{m,i} \\ &= \frac{\varpi}{2h_{m,i}} \int_{\tau_{m,i-1}}^{\tau_{m,i}} (2 - E_2\left(\tau-\tau_{n,j-1}\right) - E_2\left(\tau_{n,j}-\tau\right)) d\tau \\ &\qquad \text{if } \tau_{n,j-1} \le \tau_{m,i-1} \le \tau_{m,i} \le \tau_{n,j}; \end{aligned}$$

$$\begin{aligned} D(i,j) = & \\ &= \frac{\varpi}{2h_{m,i}} (-E_3\left(\tau_{n,j}-\tau_{m,i}\right) + E_3\left(\tau_{n,j}-\tau_{m,i-1}\right) + E_3\left(\tau_{n,j-1}-\tau_{m,i}\right) \\ &\qquad -E_3\left(\tau_{n,j-1}-\tau_{m,i-1}\right)) \\ &\qquad \text{if } \tau_{m,i-1} \le \tau_{m,i} \le \tau_{n,j-1} \le \tau_{n,j} \\ &= \frac{\varpi}{2h_{m,i}} (-E_3\left(\tau_{m,i}-\tau_{n,j}\right) + E_3\left(\tau_{m,i-1}-\tau_{n,j}\right) + E_3\left(\tau_{m,i}-\tau_{n,j-1}\right) \\ &\qquad -E_3\left(\tau_{m,i-1}-\tau_{n,j-1}\right)) \\ &\qquad \text{if } \tau_{n,j-1} \le \tau_{n,j} \le \tau_{m,i-1} \le \tau_{m,i} \\ &= \varpi(1 + \frac{1}{2h_{m,i}} (-E_3\left(\tau_{n,j}-\tau_{m,i}\right) + E_3\left(\tau_{n,j}-\tau_{m,i-1}\right) \\ &\qquad +E_3\left(\tau_{m,i}-\tau_{n,j-1}\right) - E_3\left(\tau_{m,i-1}-\tau_{n,j-1}\right))) \\ &\qquad \text{if } \tau_{n,j-1} \le \tau_{m,i-1} \le \tau_{m,i} \le \tau_{n,j}. \end{aligned} \tag{3.10}$$

Matrix C is obtained in a similar way.

To obtain the elements of matrices A_m or A_n we use formulae (3.9) by replacing n with m or vice-versa and use the fact that $E_3(0) = 1/2$:

$$A(i,j) = \tag{3.11}$$

$$= \frac{\varpi}{2h_{m,i}}(-E_3(\tau_{m,i} - \tau_{m,j}) + E_3(\tau_{m,i-1} - \tau_{m,j}) + E_3(\tau_{m,i} - \tau_{m,j-1}) + E_3(\tau_{m,i-1} - \tau_{m,j-1})) \qquad \text{if } i \neq j$$

$$= \varpi(1 + \frac{1}{h_{m,i}}(-E_3(h_{m,i}) - 1)) \qquad \text{if } i = j.$$

Remark (see also [2]) that

$$D = A_m P \tag{3.12}$$

where

$$P(k,j) = \begin{cases} 1\,, & q \times (j-1) + 1 \leq k \leq q \times j \\ 0\,, & otherwise \end{cases}$$

and, similarly,

$$C = RA_m \tag{3.13}$$

where

$$R(j,k) = \begin{cases} h_{m,k}/h_{n,j}\,, & q \times (j-1) + 1 \leq k \leq q \times j \\ 0\,, & otherwise \end{cases}.$$

4 Numerical Results

The parallelization of the code corresponding to the construction of matrices A_n, A_m, C, D and the codes corresponding to schemes 1, 2 and 3 was done on a Beowulf machine consisting of 22 processing nodes connected by a fast Ethernet switch (100Mbps). Each one is a Pentium III - 550 MHz with 128 MB of RAM. There is also a front-end dual Pentium III - 550 MHz with 512 MB of RAM. The operating system is Linux Slackware 7.0 and the software used for the communications between the nodes is MPI (Message Passing Interface) [13].

A cyclic distribution by columns of A_m allows scalability since the matrix is band. Matrix C, $n \times m$, is distributed by block columns and D, $m \times n$, by block rows because $n \ll m$. The coarse grid will have enough points to ensure convergence of the iterative refinement but it will be as small as possible so that the solution of the small linear system (2.8) can be done in all processors at a small cost in computing time.

In [6] the building of matrices C and D was done by formulae $C = RA_m$ and $D = A_m P$ to avoid some approximate integral computations of function E_3. For parallel processing, however, matrix D must be computed by (3.10) because that can be done in

Table 1. Number of iterations and elapsed times for the three iterative schemes with different number of processors ($m = 10000$, $n = 1000$)

nb. processors	Scheme 1	Scheme 2	Scheme 3
p	43 iterations	21 iterations	21 iterations
1	3472.7	3472.4	3472.2
2	1749.4	1749.0	1748.9
4	885.0	884.4	884. 4
5	714.1	713.6	713.6
10	375.3	375.1	375.1

Table 2. Speedup and efficiency for the three iterative schemes up to 10 processors ($m = 10000$, $n = 1000$)

nb. processors	Scheme 1-3	
p	S_p	E_p
1	1	1.00
2	1.99	0.99
4	3.93	0.98
5	4.86	0.97
10	9.26	0.93

Table 3. Number of iterations and elapsed times for the three iterative schemes with 10 and 20 processors ($m = 100000$, $n = 5000$)

nb. processors	Scheme 1	Scheme 2	Scheme 3
p	48 iterations	24 iterations	24 iterations
10	32590.6	32585.8	32585.1
20	14859.7	14855.0	14854.2

parallel and on the other hand the communications involved in the product $D = A_m P$ are avoided. The numerical tests performed were designed to answer the question of whether it is better to compute D and C by (3.12) and (3.13) or by (3.10). The question arises because the computation of D by (3.12) requires to many communications since A_m is distributed. This becomes more and more important as the number of processors grows. The matrix blocks needed for $C = RA_m$ are all in the same processor so it is better to use (3.13) instead of the integral formulation.

The iterative refinement is not well suited for parallelization because it requires too many communications but that is not important since the most time consuming phase of the solution of the problem is the computation of the matrices elements and that is "embarrassingly parallel". The linear system (2.8) can be solved either by block band LU factorization [10] or by a Krylov iterative method (usually preconditioned GMRES) [12].

Usually we are interested in very large dimensions and matrix A_m does not fit in one processor so it has to be computed in a distributed way. This fact forces the matrix-vector

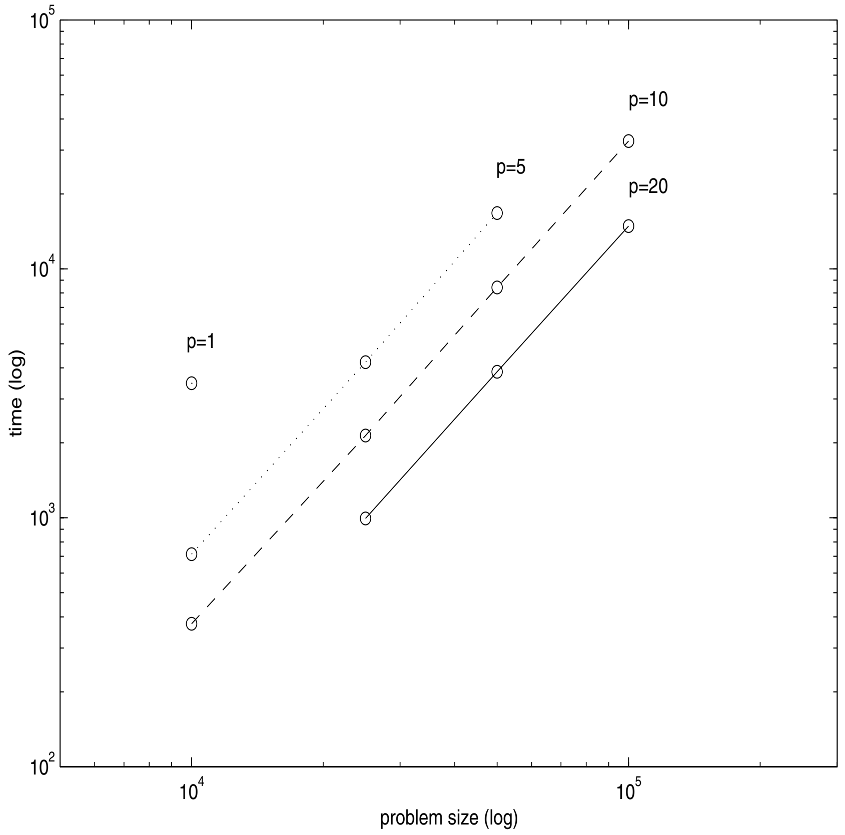


Fig. 1. Performance of the parallel code up to 20 processors

products in the refinement formulae to be done locally by blocks and the partial results have to be communicated to the other processors to be added.

The numerical tests presented correspond to $m = 10000$, $n = 1000$, $z = 1$ and the albedo $\varpi = 0.75$. A nonuniform grid was considered in $[0, \tau^*]$ by taking 4 different zones and a higher number of grid points at the beginning and at the middle of the interval. The required accuracy was residual less than 10^{-12}. We remark that for the chosen m and n values, the approximate solution for the 10000×10000 linear system is obtained without solving the corresponding system but, instead, by solving a 1000×1000 linear system and iteratively refining its solution.

Table 1 show the times of the total solution with the computation of D done by (3.10) for the three different refinement schemes. We can see that schemes 2 and 3 take approximately one half of the number of iterations of scheme 1 but the elapsed times for the three schemes are similar. Most of the time spent by the three schemes is the computation of the matrices which is similar for all of them. That explains why, as we can see in Table 2, the speedups and efficiencies are the same. As for the iterative part, schemes 2 and 3 take a little less time than scheme 1 since they require less iterations, although they perform two products by A_m instead of one. For all methods as the number of processors grows the computational elapsed time decreases.

Figure 1 shows the elapsed time per node of the solution of the problem. When the number of processors is scaled by a constant factor, the same efficiency is achieved for equidistant problem sizes on a logarithimic scale. This is denoted as isoefficiency or isogranularity, [8].

The times for the computation of D by formulae (3.10), for this example, is approximately 31 seconds for 10 processors while with formulae (3.12) it would be approximately 70 seconds. These results show that in a cluster of processors as this one, it is useful to compute D by the integral formulae because although it requires much more arithmetic it does not involve communications. This will be more important when the number of processors and their processing speed grow significantly.

With this approach we were able to solve on this machine a 100000×100000 problem based on the solution of a 5000×5000 linear system of equations, Table 3.

References

1. M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions*. Dover, New York, 1960.
2. M. Ahues, F.D. d'Almeida, A. Largillier, O. Titaud and P. Vasconcelos. An L^1 Refined Projection Approximate Solution of the Radiation Transfer Equation in Stellar Atmospheres. *J. Comput. Appl. Math.*, 140:13–26, 2002.
3. M. Ahues, A. Largillier and B.V. Limaye. *Spectral Computations with Bounded Operators*. Chapman and Hall, Boca Raton, 2001.
4. M. Ahues, A. Largillier and O. Titaud. The Roles of a Weak Singularity and the Grid Uniformity in the Relative Error Bounds. *Numer. Funct. Anal. and Optimiz.*, 22:7&8 789–814, 2001.
5. M. Ahues, F.D. d'Almeida, A. Largillier, O. Titaud and P. Vasconcelos. Iterative Refinement Schemes for an ill-Conditioned Transfer Equation in Astrophysics. *Algorithms for Approximation IV*, Univ. of Huddersfield, 70–77, 2002.
6. F.D. d'Almeida and P.B. Vasconcelos. A Parallel Implementation of the Atkinson Algorithm for Solving a Fredholm Equation. *Lecture Notes in Computer Science*, 2565:368–376, 2003.
7. K.E. Atkinson. *A Survey of Numerical Methods for the Solution of Fredholm Integral Equations of the Second Kind*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1976.
8. L.S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.
9. H. Brakhage. Uber die Numeriche Bechandlung von Integralgleichungen nach der Quadraturformelmethod. *Numer. Math.*, 2:183–196, 1960.
10. J.J. Dongarra, I.S. Duff, D.C. Sorensen and H.A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. Society for Industrial and Applied Mathematics, Philadelphia, 1998.
11. B. Rutily. Multiple Scattering Theoretical and Integral Equations. *Integral Methods in Science and Engineering: Analytic and Numerical Techniques*, Birkhauser, 211-231, 2004.
12. Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, 1996.
13. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J.J. Dongarra. *MPI: The Complete Reference*. The MIT Press, 1996.

Performance Evaluation and Design of Hardware-Aware PDE Solvers: An Introduction

Organizers: Frank Hülsemann and Markus Kowarschik

System Simulation Group, Computer Science Department
Friedrich-Alexander-University Erlangen-Nuremberg, Germany
{Frank.Huelsemann,Markus.Kowarschik}@cs.fau.de

1 Scope

In an ideal situation, all performance optimization of computationally intensive software would take place automatically, allowing the researchers to concentrate on the development of more efficient algorithms (in terms of computational complexity) rather than having to worry about performance. However, for the time being, optimizing compilers are unable to synthesize long chains of complicated code transformations to optimize program execution. As a consequence, the need to identify and to remove the performance bottlenecks of computationally intensive codes remains.

As an example of a class of computationally intensive problems, this minisymposium concentrated on the numerical solution of partial differential equations (PDEs). As with every computer program, the run times of PDE solvers depend both on the algorithms and on the data structures used in the implementations. In the context of numerical PDEs, algorithms with optimal asymptotic complexity are known for certain types of problems; e.g., multigrid methods for elliptic problems. In those cases where the optimal algorithms are applicable, only the data structures and the implementation details offer scope for improvement.

The objective of this minisymposium was to bring together researchers who are developing efficient PDE algorithms and implementations as well as researchers focusing on performance evaluation tools to predict, guide, and quantify the development of efficient numerical software.

Hardware-efficient codes in high performance simulation must follow two fundamental design goals; parallelism and locality. The idea must be to use all parallel processing units as efficiently as possible and, in addition, to optimize the performance on each individual node in the parallel environment. In particular, this requires extensive tuning for memory hierarchies, i.e., caches.

2 Outline

As was intended by the organizers, the contributions to the minisymposium cover a wide spectrum of topics. Günther et al. discuss the application of space-filling curves in order to design cache-efficient multilevel algorithms. Hülsemann et al. present an approach that combines the geometric flexibility of unstructured meshes with the relatively high processing speed of regular grid structures. The paper by Johansson et al. focuses on the analysis of the memory behavior of various PDE codes through simulation. Kowarschik et al. introduce a novel patch-adaptive and cache-aware grid processing strategy to be employed in the context of multigrid algorithms. Quinlan's contribution concentrates on a software framework for the generation of optimizing preprocessors for C++ codes. Teresco et al. discuss load balancing issues for parallel PDE solvers. Weidendorfer et al. introduce techniques for the optimization of the cache performance of iterative solvers that are based on hardware prefetching capabilities of the underlying architecture.

A Cache-Aware Algorithm for PDEs on Hierarchical Data Structures

Frank Günther, Miriam Mehl, Markus Pögl, and Christoph Zenger

Institut für Informatik, TU München
Boltzmannstraße 3, 85748 Garching, Germany
{guenthef,mehl,poegl,zenger}@in.tum.de

Abstract. A big challenge in implementing up to date simulation software for various applications is to bring together highly efficient mathematical methods on the one hand side and an efficient usage of modern computer archtitectures on the other hand. We concentrate on the solution of PDEs and demonstrate how to overcome the hereby occuring quandary between cache-efficiency and modern multilevel methods on adaptive grids. Our algorithm is based on stacks, the simplest possible and thus very cache-efficient data structures.

1 Introduction

In most implementations, competitive numerical algorithms for solving partial differential equations cause a non-negligible overhead in data access and, thus, can not exploit the high performance of processors in a satisfying way. This is mainly caused by tree structures used to store hierarchical data needed for methods like multi-grid and adaptive grid refinement.

We use space-filling curves as an ordering mechanism for our grid cells and – based on this order – to replace the tree structure by data structures which are processed linearly. For this, we restrict to grids associated with space-trees (allowing local refinemt) and (in a certain sense) local difference stencils. In fact, the only kind of data structures used in our implementation is a fixed number of stacks. As stacks can be considered as the most simple data structures used in Computer Science allowing only the two basic operations push and pop[1], data access becomes very fast – even faster than the common access of non-hierarchical data stored in matrices – and, in particular, cache misses are reduced considerably. Even the implementation of multi-grid cycles and/or higher order discretizations as well as the parallelization of the whole algorithm becomes very easy and straightforward on these data structures and doesn't worsen the cache efficiency.

In literature, space-filling curves are a well-known device to construct efficient grid partitionings for data parallel implementations of the numerical solution of partial differential equations [23,24,13,14,15,16,19]. It is also known that – due to locality properties of the curves – reordering grid cells according to the numbering induced by a space-filling curve improves cache-efficiency (see e.g. [1]). Similar benefits of reordering data along space-filling curves can also be observed for other applications like e.g. matrix transposition [5] or matrix multiplication [6]. We enhance this effect by constructing

[1] push puts data on top of a pile and pop takes data from the top of a pile.

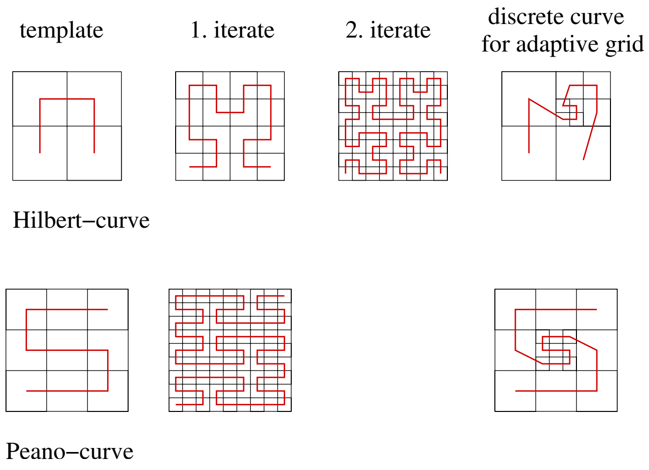


Fig. 1. Generating templates, first iterates and discrete curve on an adaptively refined grid for two-dimensional Hilbert- and Peano-curves

stacks for which we can even *completely avoid* ‘jumps’ in the adresses instead of only reducing their probability or frequency.

2 Space-Filling Curves and Stacks

As we look for an odering mechanism for the cells of a multilevel adaptive rectangular grid based on a space tree, we restrict our attention to a certain class of space-filling curves[2], namely recursively defined, self-similar space-filling curves with rectangular recursive decomposition of the domain. These curves are given by a simple generating template and a recursive refinement procedure which describes the (rotated or mirrored) application of the generating template in sub-cells of the domain. See figure 1 for some iterates of the two-dimensional Hilbert- and Peano-curves, which are prominent representatives of this class of curves. As can be seen, the iterate we use in a particular part of our domain depends on the local resolution.

Now, these iterates of the space-filling curves associated to the given grid – also called discrete space-filling curves – and not the space-filling curve itself[3] define the processing order of grid cells. To apply an operator-matrix to the vector of unknowns, we process the cells strictly in this order and perform all computations in a cell-oriented way, which means that in each cell, we evaluate only those parts of the corresponding operator that can be computed solely with the help of the cell’s own information and, in particular, without adressing any information of neighbouring cells. This method is standard for finite element methods (see e.g. [4]), but can be generalized for ‘local’

[2] For general information on space-filling curves see [20].

[3] The space-filling curve itself can be interpreted as the limit of the recursive refinement procedure and is – in contrast to the iterates – *not* injective.

discrete operators, which means that for the evaluation in one grid point, only direct neighbours are needed[4].

To get more concrete, we look at a space-tree with function values located at the vertices of cells in the following. To apply the common five-point-stencil for the Laplacian operator, for example, we have to decompose the operator into four parts associated to the four neighbouring cells of a gridpoint[5]:

$$\Delta_h u_h|_{i,j} =$$

$$\frac{u_{i-1,j}+u_{i,j-1}-2u_{i,j}}{2h^2}+\frac{u_{i+1,j}+u_{i,j-1}-2u_{i,j}}{2h^2}+\frac{u_{i+1,j}+u_{i,j+1}-2u_{i,j}}{2h^2}+\frac{u_{i-1,j}+u_{i,j+1}-2u_{i,j}}{2h^2}.$$

$$\begin{array}{ccccc} \cdot & - & \cdot & - & \cdot \\ | & & | & & | \\ \frac{1}{2} & - & -1 & - & \cdot \\ | & & | & & | \\ \cdot & - & \frac{1}{2} & - & \cdot \end{array} \qquad \begin{array}{ccccc} \cdot & - & \cdot & - & \cdot \\ | & & | & & | \\ \cdot & - & -1 & - & \frac{1}{2} \\ | & & | & & | \\ \cdot & - & \frac{1}{2} & - & \cdot \end{array} \qquad \begin{array}{ccccc} \cdot & - & \frac{1}{2} & - & \cdot \\ | & & | & & | \\ \cdot & - & -1 & - & \frac{1}{2} \\ | & & | & & | \\ \cdot & - & \cdot & - & \cdot \end{array} \qquad \begin{array}{ccccc} \cdot & - & \frac{1}{2} & - & \cdot \\ | & & | & & | \\ \frac{1}{2} & - & -1 & - & \cdot \\ | & & | & & | \\ \cdot & - & \cdot & - & \cdot \end{array}$$

Of course, in each cell, we evaluate the cell-parts of the operator values for all four vertices all at once. Thus, we have to construct stacks such that all data associated to the respective cell-vertices lie on top of the stacks when we enter the cell during our run along the discrete space-filling curve. We will start with a simple regular two-dimensional grid illustrated in figure 2 and nodal data. If we follow the Peano-curve, we see that during our run through the lower part of the domain, data points on the middle line marked by 1 to 9 are processed linearly from the left to the right, during the subsequent run through the upper part vice versa from the right to the left. Analogously, all other grid points can be organized on lines which are processed linearly forward and, afterwards, backward. For the Peano-curve and a regular grid, this can be shown to hold for arbitrarily fine grids, as well.

As these linearly processed lines work with only two possibilities of data access, namely push (write data at the next position in the line) and pop (read data from the actual position of the line), they correspond directly to our data stacks. We can even integrate several lines in one stack, such that it is sufficient to have two stacks: $1D_{black}$ for the points on the right-hand-side of the curve and $1D_{red}$ for the points at the left-hand-side of the curve. The behavior of a point concerning read and write operations on stacks can be predicted in a deterministic[6] way. We only have to classify the points as 'inner points' or 'boundary points' and respect the (local) progression of the curve.

[4] In addition, in the case of adaptively refined grid, the cell-oriented view makes the storage of specialized operators at the boundary of local refinements redundant as a single cell computes its contribution to the overall operators without taking into consideration the refinement depth of surrounding cells.

[5] If we want to minimze the number of operations, this equal decomposition is of course not the optimal choice, at least not for equidistant grids and constant coefficients.

[6] In this context, deterministic means depending only on locally available informations like the local direction of the Peano-curve.

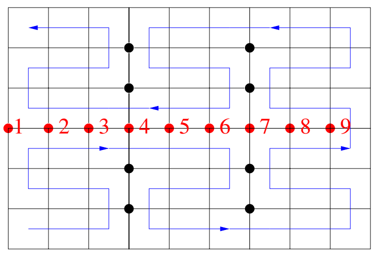


Fig. 2. Example in 2D using the Peano-curve

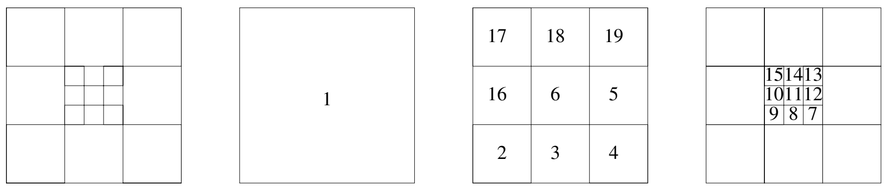


Fig. 3. Example for the order of hierarchical cells defined by the discrete Peano-curve

We use the Peano-curve instead of the Hilbert-curve since in 3D the Peano-curve guarantees the inversion of the processing order of grid points along a plane in the domain if we switch from the domain on one side of the plane to the domain on the other side. This property is essential for our stack concept and we could not find any Hilbert-curve fulfilling it. There are good reasons to assume that this is not possible at all.

Up to this point, we have restricted our considerations to regular grids to explain the general idea of our algorithm, but the whole potential of our approach shows only when we look at adaptively refined grids and – in the general case – hierarchical data in connection with generating systems [11]. This leads to more than one degree of freedom per grid point and function. Even in this case, the space-filling curve defines a linear order of cells respecting all grid levels: the cells are visited in a a top-down depth-first process reflecting the recursive definition of the curve itself (see figure 3 for an example).

In our stack-context, we have to assure that even then predictable and linear data access to and from stacks is possible. In particular, we have to assure that grid points of different levels lie on the stacks in the correct order and do not “block” each other. We could show that it is sufficient to use four colors representing four different stacks of the same type and a second type of stack, called $0D$ stack. Thus, we end up with a still fixed – and in particular independent of the grid resolution – number of eight stacks (for details see [10]).

To be able to process data on a domain several times – as needed for each iterative solver – without loss of efficiency, we write all points to a so called 2D- or plain-stack as soon as they are ‘ready’. It can easily be seen that, if we process the grid cells in opposite direction in the next iteration, the order of data within this 2D-stack enables us to pop all grid points from this 2D-stack as soon as they are ‘touched’ for the first time.

Such, we can use the 2D-stack as input stack very efficiently. We apply this repeatedly and, thus, change the processing direction of grid cells after each iteration.

An efficient algorithm deducted from the concepts described above passes the grid cell-by-cell, pushes/pops data to/from stacks deterministically and automatically and will be cache-aware by concept (not by optimization!)[7] because of the fact, that using linear stacks is a good idea in conjunction with modern processors prefetching techniques. An additional gain of using our concept is the minimal storage cost for administrational data. We can do completely without any pointer to neighbours and/or father- and son-cells. Instead, we only have to store one bit for refinement informations and one bit for geometrical information (in- or outside the computational domain) per cell.

3 Results

To point out the potential of our algorithm, we show some first results for simple examples with the focus on the efficiency concerning storage requirements, processing time and cache behavior. Note that up to now we work with an experimental code which is not optimized at all yet. Thus, absolute values like computing time will be improved further and our focus here is only on the cache behavior and the qualitative dependencies between the number of unknowns and the performance values like computing times.

3.1 Two-Dimensional Poisson Equation

As a first test, we solve the two-dimensional Poisson equation on the unit-square and on a two-dimensional disc with homogeneous Dirichlet boundary conditions:

$$\triangle\, u(\mathbf{x}) = -2\pi^2 \sin(\pi x)\sin(\pi y), \quad \forall\, \mathbf{x} = (x,y)^T \in \Omega \tag{3.1}$$

$$u(\mathbf{x}) = \sin(\pi x)\cdot\sin(\pi y) \quad \forall\, \mathbf{x} \in \partial\Omega \tag{3.2}$$

The exact solution of this problem is given by $u(\mathbf{x}) = \sin(\pi x)\cdot\sin(\pi y)$. To discretize the Laplacian operator, we use the common Finite Element stencil arising from the use of bilinear basis functions. The resulting system of linear equations was solved by an additive multi-grid method with bilinear interpolation and full-weighting as restriction operator. As criterion for termination of the iteration loop we took a value of $r_{max} = 10^{-5}$, where r_{max} is the maximum (in magnitude) of all corrections over all levels.

On the unit square, we used regular grids with growing resolution. On the disc, we used different adaptive grids gained by local coarsening strategies starting from an initial grid with 729×729 cells. To get a sufficient accuracy near the boundary, we did not allow any coarsening of boundary cells. Tables 1 and 2 show performance values obtained on a dual Intel XEON 2.4 GHz with 4 GB of RAM.

Note that the number of iterations until convergence is in both cases independent from the resolution, which one would have expected for a multi-grid method. The analysis of cache misses and cache hits on the level 2 cache gives a good hint on the high efficiency of our method. But for a more realistic judgement of the efficieny, the very good relation

[7] Algorithms wich are cache-aware by concept without detailed knowledge of the cache parameters are also called cache-oblivious [18,9,8].

Table 1. Performance values for the two-dimensional Poisson equation on the unit square solved on a dual Intel XEON 2.4 GHz with 4 GB of RAM

resolution	iterations	L2 cache misses per it.	rate
243×243	39	134,432	1.09
729×729	39	1,246,949	1.12
2187×2187	39	11,278,058	1.12

Table 2. Performance values for the two-dimensional Poisson equation on a disk solved on a dual Intel XEON 2.4 GHz with 4 GB of RAM

variables	% of full grid	L2-hitrate	iter	cost per variable
66220	15.094%	99.28%	287	0.002088
101146	23.055%	99.26%	287	0.002031
156316	35.630%	99.24%	286	0.001967
351486	80.116%	99.11%	280	0.001822
438719	100.00%	99.16%	281	0.002030

between the number of actual cache-misses and the minimal number of necessary cache-misses caused by the unavoidable first loading of data to the cache is more significant:

With the size s of a stack element, the number n of degrees of freedom and the size cl of a L2 cache line on the used architecture we can guess a minimum $cm_{min} = \frac{n \cdot s}{cl}$ of cache misses per iteration, which has to occur if we read each grid point once per iteration producing $\frac{s}{cl}$ cache misses per point. In fact, grid points are typically used four times in our algorithm as well as in most FEM-algorithms. Thus, this minimum guess is assumed to be even too low. 'Rate' in table 1 is defined as $\frac{cm_{real}}{cm_{min}}$ where cm_{real} are the L2 cache misses per iteration simulated with `calltree` [25]. As this rate is nearly one in our algorithm, we can conclude that we produce hardly no needless cache misses at all. In addition, we get a measured L2-hit-rate of at least 99,13%, using hardware performance counters [26] on an Intel XEON, which is a very high value for 'real' algorithms beyond simple programs for testing performance of a given architecture.

In table 2, the column 'costs per variable' shows the amount of time needed per variable and per iteration. These values are nearly constant. Thus, we see that we have a linear dependency between the number of variables and the computing time, no matter if we have a regular full grid or an adaptively refined grid. This is a remarkable result as we can conclude from this correlation that, in our method, adaptivity doesn't any additional costs.

3.2 Three-Dimensional Poisson Equation

In the previous sections, we only considered the two-dimensional case. But our concepts can be generalized in a very natural way to three or even more dimensions. Here, we give a few preliminary results on the 3D case. The basic concepts are the same as in the

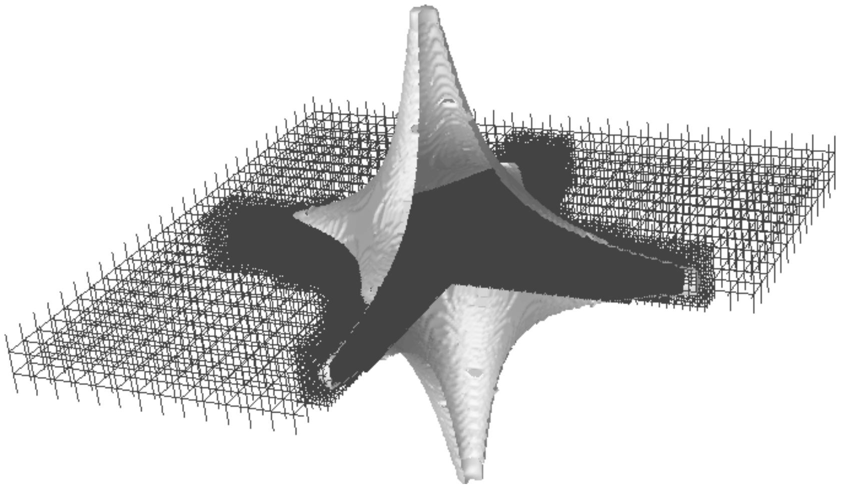


Fig. 4. Representation of the star-shaped domain with the help of adaptive grids gained by a geometry oriented coarsening of a $243 \times 243 \times 243$ grid

two-dimensional case, but to achieve an efficient implementation, we introduced some changes and/or enhancements in the concrete realization [17].

The first – and obvious – change is that we need 3D in- and output stacks and 2D, 1D and 0D stacks during our run through the space tree instead of 1D and 0D stacks only. In addition, we use 8 colors for the 0D, 12 colors for the 1D stacks, and 6 colors for the 2D stacks. Another interesting aspect in 3D is that we replace the direct refinement of a cell by the introduction of 27 sub-cells by a dimension recursive refinement: A cell is cut into three "plates" in a first step, each of these plates is cut into three "bars", and, finally, each bar into three "cubes". This reduces the number of different cases dramatically and can even be generalized to arbitrary dimensions, too.

We solve the three-dimensional Poisson equation

$$\triangle u((\mathbf{x})) = 1 \tag{3.3}$$

on a star-shaped domain (see figure 4) with homogeneous boundary conditions. The Laplace operator is discretized by the common Finite Difference stencil and – analoguously to the 2D case – we use an additive multi-grid method with trilinear interpolation and full-weighting as restriction operator. The termination criterion was $|r_{max}| \leq 10^{-5}$.

Table 3 shows performance values obtained on a dual Intel XEON 2.4 GHz with 4 GB of RAM for adaptively refined grids. As can be seen, all important performance properties like multi-grid performance and cache efficiency carry over from the 2D case. The rate between the number of real cache misses and the minimal number of expected cache misses was measured for a different example (poisson equation in a sphere) and stays also in the range of 1.10. Thus, even in 3D, the essential part of the cache misses is really necessary and cannot be avoided by any algorithm.

Table 3. Performance values for the three-dimensional Poisson equation on a star-shaped domain solved on a dual Intel XEON 2.4 GHz with 4 GB of RAM

max. resolution	variables	storage req.	L2-hit-rate	iter	cost per variable
$81 \times 81 \times 81$	18,752	0.7MB	99.99%	96	0.0998
$243 \times 243 \times 243$	508,528	9MB	99.99%	103	0.0459
$729 \times 729 \times 729$	13,775,328	230MB	99.98%	103	0.0448

4 Conclusion

In this paper we presented a method combining important mathematical features for the numerical solution of partial differential equations – adaptivity, multi-grid, geometrical flexibility – with a very cache-efficient way of data organization tailored to modern computer architectures and suitable for general discretized systems of partial differential equations (yet to be implemented). The parallelization of the method follows step by step the approach of Zumbusch [23] who has already used successfully space-filling curves for the parallelization of PDE-solvers. The generalization to systems of PDEs is also straightforward. With an experimental version of our programs we already solved the non-stationaray Navier-Stokes equations in two dimensions, but the work is still in progress and shall be published in the near future. The same holds for detailed performance results for various computer architectures.

References

1. M.J. Aftosmis, M.J. Berger, and G. Adomavivius. A Parallel Multilevel Method for adaptively Refined Cartesian Grids with Embedded Boundaries. AIAA Paper, 2000.
2. R.A. Brualdi and B.L. Shader. On sign-nonsingular matrices and the conversion of the permanent into the determinant. In P. Gritzmann and B. Sturmfels, editors. Applied Geometry and Discrete Mathematics, *The Victor Klee Festschrift*, pages 117-134, Providence, RI, 1991. American Mathematical Society.
3. F.A. Bornemann. An adaptive multilevel approach to parabolic equations III: 2D error estimation and multilevel preconditioning. *IMPACT Computational Science and Engeneering*, 4:1-45, 1992.
4. Braess. Finite Elements. Theory, Fast Solvers and Applications in Solid Mechanics. Cambridge University Press, 2001.
5. S. Chatterjee and S. Sen. Chache-Efficient Matrix Transposition. In *Proceedings of HPCA-6*, pages 195–205, Toulouse, France, January 2000.
6. S. Chatterjee, A.R. Lebeck, P.K. Patnala, and M. Thottethodi. Recursive array layouts and fast parallel matrix multiplication. In *Proceedings of Eleventh Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 222-231, Saint-Malo, France, 1999.
7. W. Clarke. Key-based parallel adaptive refinement for FEM. Bachelor thesis, Australian National Univ., Dept. of Engineering, 1996.
8. E.D. Demaine. Cache-Oblivious Algorithms and Data Structures. In *Lecture Notes from the EEF Summer School on Massive Data Sets*, Lecture Notes in Computer Science, BRICS, University of Aarhus, Denmark, June 27-July 1, 2002, to appear.

9. M. Frigo, C.E. Leierson, H. Prokop, and S. Ramchandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual Sympoisium on Foundations of Computer Science*, pages 285-297, New York, October 1999.
10. F. Günther. Eine cache-optimale Implementierung der Finite-Elemente-Methode. Doctoral thesis, Institut für Informatik, TU München, 2004.
11. M. Griebel. Multilevelverfahren als Iterationsmethoden über Erzeugendensystemen. Habilitationsschrift, TU München, 1993.
12. M. Griebel, S. Knapek, G. Zumbusch, and A. Caglar. Numerische Simulation in der Moleküldynamik. *Numerik, Algorithmen, Parallelisierung, Anwendungen*, Springer, Berlin, Heidelberg, 2004.
13. M. Griebel and G.W. Zumbusch. Parallel multigrid in an adaptive PDE solver based on hashing and space-filling curves. *Parallel Computing*, 25:827-843, 1999.
14. M. Griebel and G. Zumbusch. Hash based adaptive parallel multilevel methods with space-filling curves. In H. Rollnik and D. Wolf, editors, *NIC Series*, 9:479-492, Germany, 2002. Forschungszentrum Jülich.
15. J.T. Oden, A. Para, and Y. Feng. Domain decomposition for adaptive *hp* finite element methods. In D.E. Keyes and J. Xu, editors, Domain decomposition methods in scientific and engineering computing, *proceedings of the 7th int. conf. on domain decomposition*, vol. 180 of Contemp. Math., pages 203-214, 1994, Pennsylvania State Universitiy.
16. A.K. Patra, J. Long, A. Laszloff. Efficient Parallel Adaptive Finite Element Methods Using Self-Scheduling Data and Computations. HiPC, pages 359-363, 1999.
17. M. Pögl. Entwicklung eines cache-optimalen 3D Finite-Element-Verfahrens für große Probleme. Doctoral thesis, Institut für Informatik, TU München, 2004.
18. H. Prokop. Cache-Oblivious Algorithms. Master Thesis, Massachusetts Institute of Technology, 1999.
19. S. Roberts, S. Klyanasundaram, M. Cardew-Hall, and W. Clarke. A key based parallel adaptive refinement technique for finite element methods. In B.J. Noye, M.D. Teubner, and A.W. Gill, editors, *Proc. Computational Techniques and Applications: CTAC '97*, pages 577-584, World Scientific, Singapore, 1998.
20. H. Sagan. Space-Filling Curves. Springer-Verlag, New York, 1994.
21. R.J. Stevens, A.F. Lehar, and F.H. Preston. Manipulation and Presentation of Multidimensional Image Data Using the Peano Scan. *IEEE Trans. Pattern An. and Machine Intelligence*, Vol PAMI-5, pages 520-526, 1983.
22. L. Velho, J. de Miranda Gomes. Digital Halftoning with Space-Filling Curves. *Computer Graphics*, 25:81-90, 1991.
23. G.W. Zumbusch. Adaptive Parallel Multilevel Methods for Partial Differential Equations. Habilitationsschrift, Universität Bonn, 2001.
24. G.W. Zumbusch. On the quality of space-filling curve induced partitions. *Z. Angew. Math. Mech.*, 81:25-28, 2001. Suppl. 1, also as report SFB 256, University Bonn, no. 674, 2000.
25. J. Weidendorfer, M. Kowarschik, and C. Trinitis. A Tool Suite for Simulation Based Analysis of Memory Access Behavior, *Proceedings of the 2004 International Conference on Computational Science, Krakow, Poland, June 2004*, vol. 3038, Lecture Notes in Computer Science (LNCS), Springer.
26. `http://user.it.uu.se/ mikpe/linux/perfctr/`

Constructing Flexible, Yet Run Time Efficient PDE Solvers[⋆]

Frank Hülsemann and Benjamin Bergen

System Simulation Group, Computer Science Department
Friedrich-Alexander-University Erlangen-Nuremberg, Germany
{frank.huelsemann,ben.bergen}@cs.fau.de

Abstract. Amongst other properties, PDE solvers for large scale problems should be flexible, as they are time consuming to write, and obviously run time efficient. We report on the experiences with a regularity centred approach for grid based PDE software that aims to combine geometric flexibility with run time efficiency. An unstructured coarse grid that describes the problem geometry is repeatedly subdivided in a regular fashion to yield a hierarchy of grids on which the approximation is sought. By construction, the grid hierarchy is well suited for multilevel methods. The gain in run time performance that results from the exploitation of the patch wise regularity of the refined grids over standard implementations will be illustrated.

1 Introduction

Two desirable properties of PDE solvers, flexibility and run time efficiency, are often seen as difficult to combine. In this article we discuss the optimisation potential for the implementation that is inherent to sparse linear system solvers on regularly refined, unstructured grids which commonly arise in geometric multigrid methods. Although the developed techniques apply to any two or higher dimensional problem, we concentrate on two and three spatial dimensions. The run time behaviour of a PDE solver for a given problem size and a given hardware platform depends on the algorithmic complexity of the solution method employed and on its implementation. Therefore, in order to be fast, an efficient PDE solver will have to combine an optimal algorithm with an appropriate implementation. For our discussion, we concentrate on geometric multigrid methods as solvers, which are known to be of optimal linear algorithmic complexity for certain second order, elliptic boundary value problems [4], [7].

The type of flexibility that we are interested in concerns the use of unstructured grids. In this work, we do not consider other, undoubtedly desirable, aspects of solver flexibility such as the availability of different approximation schemes, for example higher order basis functions or staggered grids to name but two, nor do we cover adaptive methods.

Our main concern is to improve the floating point performance of geometric multigrid methods on unstructured grids. For this purpose, it suffices to consider cases for which (standard) geometric multigrid methods are applicable. Hence we do not consider the

[⋆] This work has been supported by a KONWIHR grant of the Bavarian High Performance Computing Initiative.

question of how to construct robust multigrid schemes on unstructured grids with high aspect ratios or for problems with strongly varying coefficients.

The article is structured as follows. First, we give reasons why repeated refinement is a common approach in the implementation of geometric multigrid methods on unstructured grids. Then, in Sect. 3, we identify the different types of regularity in the discretisation that can be exploited. After that, we present the data structures that capture the regular properties in Sect. 4. Section 5 accesses the cost of the data synchronisation that are inherent to our approach. Following on, we present performance data from numerical experiments before we conclude the paper.

2 Geometric Multigrid on Unstructured Grids

The common approach to generating the hierarchy of nested approximation spaces needed for standard geometric multigrid on unstructured grids consists in carrying out several steps of repeated regular refinement. The reason for the popularity of this approach lies in the complexity and difficulty of any alternative. Coarsening a given unstructured grid so that the resulting approximation spaces are nested is often simply impossible. Hence, one would have to work on non-nested spaces. Multigrid methods for such a setting are much more complicated than those for their nested counterparts. In summary, although it is possible to construct geometric multigrid methods on hierarchies of repeatedly coarsened unstructured grids, this option is rarely chosen in practice. Hence we turn our attention to the standard approach of repeated refinement.

The basic assumption underlying this strategy is that the spatial resolution of the unstructured input grid is not high enough in any section of the grid (anywhere) to yield a sufficiently accurate solution. In such a case, some global refinement is indeed necessary. While this assumption excludes the cases in which the geometry is so complex that the resolution of the domain geometry is already sufficient for the accuracy requirements, it still includes various challenging applications that require large scale computations. Applications with high accuracy requirements on comparatively simple domains arise for example in the simulation of plasma in particle accelerators, density functional computations in molecular dynamics and the simulation of wave propagation in high energy ultra sound.

3 Identifying Regularity on Globally Unstructured Grids

We illustrate the main concepts of our approach using a simplified problem domain that arose in an electromagnetic field problem. Consider the setting in Fig. 1. For ease of presentation, we assume a Poisson problem

$$\begin{aligned} -\triangle u &= f \qquad \text{on } \Omega \\ u &= 0 \qquad \text{on } \partial\Omega. \end{aligned}$$

on the problem domain Ω. The extension of the implementation techniques to other settings is straightforward.

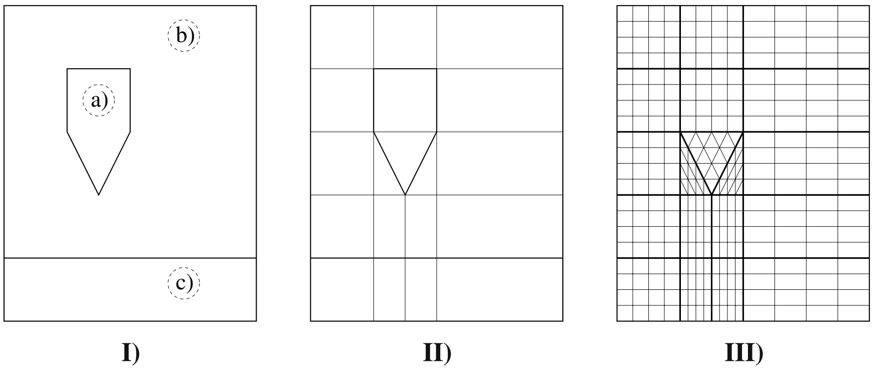


Fig. 1. Non-destructive testing example, from left to right: **I)** the problem domain with a) the coil that generates a magnetic field, b) air surrounding the coil and c) the material to be tested. The material parameters (magnetic permeability) of the three different components differ from each other but are assumed to be constant within each component (homogeneous materials). **II)** the coarse, unstructured grid that represents the problem geometry and **III)** the grid after two global regular subdivision steps

For the representation of the discrete operator on the grid in Fig. 1 II), it is appropriate to employ sparse matrix storage schemes. For a description of various sparse matrix formats and their implementation see [1], for example. Returning to the example, we note that the grid consists of different cell geometries (triangles and quadrilaterals), the number of neighbours is not constant for all vertices and the material parameters vary. However, the situation on the grid in Fig. 1 III) is different. Within each cell of the unstructured input grid the repeated refinement has resulted in regular patches. In the interior of each coarse grid cell the grid is regular and the material parameter is constant. This implies that one stencil suffices to represent the discrete operator inside such a regular region[1]. Put differently, if the stiffness matrix was set up for the refined grid, then, assuming an appropriate numbering of the unknowns, the row entries belonging to each patch would display a banded structure with constant matrix coefficients. From this observation it is clear that once the patches contain sufficiently many points it will not be desirable to ever set up a sparse matrix representation of the discrete operator. The unstructured and hybrid nature of the coarse input grid shows itself in the fine grid representation of the discrete operator at the vertices and at the points on the edges of the coarse grid. The vertices of the input grid have to be treated in a standard, unstructured manner, because there is no regularity to be exploited at these points. In two spatial dimensions, the fine grid points that resulted from the refinement of an edge of the input grid share a common stencil. The stencil shape and the stencil entries can vary from edge to edge, depending on the geometric shapes of the cells that share the edge, but along one edge, both shape and entries are constant. In three spatial dimensions, this

[1] In the description of the grid structure, we use the terms (regular) patch and region interchangeably.

observation holds for the faces whereas the (points on the) edges have to be treated in an unstructured way.

In summary, within each patch in Fig. 1 III) we have

– a constant stencil shape (banded matrix structure) for the discrete operator,
– constant material parameters and therefore
– constant stencil entries.

Next we will describe the data structures that we propose to exploit the regular structure of the operator within each patch to improve the floating point performance of the geometric multigrid implementation on unstructured input grids and give parallel performance results on different architectures. Further descriptions of the approach can be found in [2], [3], [5].

4 Data Structures for Patch Wise Regularity

As pointed out before, a general sparse matrix storage format is not suitable for the adequate representation of the patch wise structure of the discrete operator on a highly refined grid.

The main idea of our approach is to turn operations on the refined, globally unstructured grid into a collection of operations on block structured parts where possible and resort to unstructured operations only where necessary. As described above, in two spatial dimensions only the vertices of the input grid require inherently unstructured operations, while each edge and each computational cell of the input grid constitute a block structured region. For the remainder of the paper, we concentrate on the case where in each block structured part, the stencil shape and the stencil entries are constant, but the shape and the entries can of course differ between parts. The storage scheme is illustrated in Fig. 2. Each block structured region provides storage for the unknowns and the discrete operators in its interior. In addition to the memory for the degrees of freedom of one problem variable, the regular region also provides buffers for information from neighbouring objects, which can be other regular regions or unstructured regions and points. When applying a discrete operator to a variable, this rim of buffers eliminates the need to treat the unknowns close to the boundary, where the stencil extends into neighbouring objects, any different from those in the inside. In other words, the number of points to which a constant stencil can be applied in the same manner is maximised.

We mentioned before that in the regular regions, the discrete operator can be expressed by stencils with constant shape. In some cases, the entries of the stencil are also constant over the object. Inside such a region, the representation of the operator can be reduced to one single stencil. Both properties, constant stencil shape and constant stencil entries, help to improve the performance of operations involving the discrete operator. The scale of the improvement depends on the computer architecture.

First, we turn to the constant shape property. On general unstructured grids, the discrete operator is usually stored in one of the many sparse matrix formats. If one uses such general formats in operations such as the matrix-vector product or a Gauß-Seidel iteration, one usually has to resort to indirect indexing to access the required entries in the vector. Being able to represent the discrete operator in the regular regions by

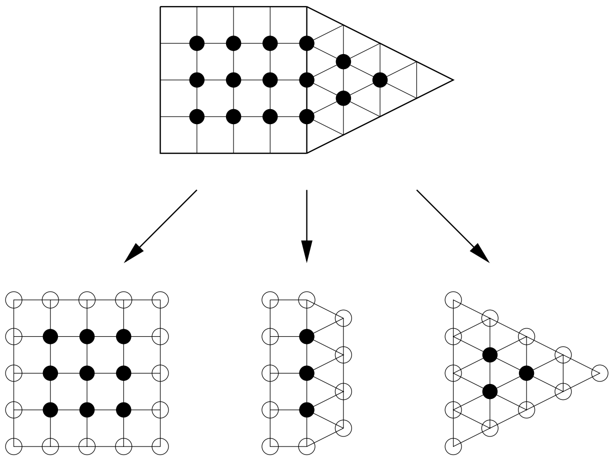


Fig. 2. Data layout for stencil based operations: Assuming a vertex based discretisation with Dirichlet boundary conditions on the top grid, the unknowns are stored in three different memory segments to allow stencil operations. Full circles indicate unknowns, hollow circles indicate buffered values from neighbouring objects (ghost cells)

stencils with fixed shapes, we can express the memory access pattern for the operation explicity through index arithmetic and thus enable the compiler, or indeed hardware mechanisms like prefetch units, to analyse and optimise the memory accesses better. In the sparse matrix case, the memory access pattern is known at run time, in our setting it is known at compile time, at least for a significant subset of all points. The size of this subset will be discussed in the section on data synchronisation costs. On the Hitachi SR8000 supercomputer at the Leibniz computing centre in Munich, this change from indirect indexing to index arithmetic improves the MFLOP/s performance of a Gauß-Seidel iteration on a single processor from around 50 MFLOP/s for a CRS (compressed row storage) implementation to 300 MFLOP/s for the stencil based computation. These values were obtained with a 27 point finite element discretisation inside a refined hexahedron on a processor with a theoretical peak performance of 1500 MFLOP/s. On this high memory bandwidth architecture, the explicit knowledge about the structure of the operations results in a six-fold performance improvement. Many cache based architectures do not offer such a high bandwidth to main memory. Given that a new stencil has to be fetched for each unknown, the memory traffic to access the stencil values slows down the computations on these machines. As an example for such machines, we consider an Intel Pentium 4 processor with 2.4 GHz clock speed, 533 MHz (effective) front side bus and dual channel memory access. Our CRS based Gauß-Seidel iteration runs at 190 MFLOP/s on a machine with a theoretical peak performance of 4800 MFLOP/s[2]. With the fixed stencil shape implementation, we observe a performance between 470 and 490 MFLOP/s, depending on the problem size, which is 2.5 times more than for the

[2] The results on the PC architecture were obtained with the version 3.3.3 of the GNU compiler collection.

standard unstructured one. In all cases, the problem size did not fit into the caches on the machines.

In the case of constant stencil entries for an element, the advantage of the structured implementation is even clearer. Instead of fetching 27 stencil values from memory for each unknown, the same stencil is applied to all unknowns in the element. This obviously reduces the amount of memory traffic significantly. On the Hitachi SR8000, such a constant coefficient Gauß-Seidel iteration achieves 884 (out of 1500) MFLOP/s on a hexahedron with 199^3 unknowns. Compared to the standard unstructured implementation, this amounts to a speed up factor of 17.5. Again, on the commodity architecture Pentium 4, the improvement is less impressive. For a problem of the same size, the constant coefficient iteration runs at 1045 MFLOP/s, which is 5.5 times faster than its unstructured counterpart.

These numbers make it clear that exploiting the substructure does improve the run time performance inside the regular regions. However, in general, the global grid consists of more than one regular block, which introduces the need for data synchronisation between these blocks. In the next section, we consider the computational cost of the communication between blocks and indicate how many levels of refinement are necessary for the proposed data structures to pay off.

5 Cost of Data Synchronisation

In the previous section it has been shown that the run time performance inside the regular regions improves when the structure of the computations is taken into account. However, when comparing the performance of a structured implementation with its unstructured counterpart, one also has to take the computational cost for the data synchronisation into account. The computations inside a block can take place once the current boundary values have been stored in the rim buffers. These boundary values have to be collected from the neighbouring grid objects. For a hexahedron, this means that it has to receive the current values from its six neighbouring faces, its 12 edges and eight corners. Furthermore, after the values of the unknowns inside the hexahedron have been updated, the neighbouring objects have to be informed about the changes, which results again in data exchange operations.

It is reasonable to assume that the time requirements for the data synchronisation within one process are linearly proportional to the number of values that have to be copied back and forth. Consequently, we consider the ratio between the number of points inside a hexahedra, for which the computations can be carried out in the optimised manner described above, and the number of points on the hexahedra boundary, which have to be exchanged. Only when the computations inside the regular region take sufficiently longer than the communication of the boundary values can the performance on the proposed data structures be higher than on the standard unstructured ones. In Table 1 we give the numbers for the different point classes for a repeatedly refined hexahedron and the ratio of the regular points (N_r) over all points (N). The refinement rule consists in subdividing a cube into eight sub-cubes of equal volume. The refinement level 0 corresponds to the initial cube, level $l, l \geq 1$ is the result of subdividing all cubes in level $l-1$ according to the rule above.

Table 1. Point numbers in a refined hexahedron (cube): l denotes the refinement level, N_r the number of points in the inside of a refined cube, N the total number of points in the cube including the boundary and $r = N_r/N$ the ratio between regular points and all points

l	N_r	N	r
0	0	8	0
1	1	27	0.037
2	27	125	0.216
3	343	729	0.471
4	3375	4913	0.687
5	29791	35937	0.829
6	250047	274625	0.911
7	2048383	2146689	0.954
8	16581375	16974593	0.977

We introduce some notation in order to estimate the overall performance on one cube with the communication taken into account. We assume that all N points can be updated with the same number of floating point operations, denoted by f. If the standard, unstructured implementation is employed to update the boundary points, we observe a floating point performance of P_{crs} MFLOP/s on these $N - N_r = (1 - r) * N$ points. In the regular region, comprising N_r points, the computations are carried out with P_{reg} MFLOP/s. The value $s = P_{reg}/P_{crs}$ is the speed up factor in the observed floating point performance. It turns out to be useful to express the communication time in terms of floating point operations that could have been performed in the same time. By m we denote the number of floating point operations that can be carried out at an execution speed of P_{reg} MFLOP/s in the same period of time that it takes to copy one point value to or away from the cube. In total, there are $(1 - r)N$ points on the boundary and for simplicity we assume that the number of values to be copied from the hexahedra to its neighbours is also $(1 - r)N$, which is an upper bound on the actual number. Thus we arrive at $2(1 - r)N$ copy operations, each one taking m/P_{reg} seconds, resulting in a communication time t_{syn} of $t_{syn} = \frac{2(1-r)N\times m}{P_{reg}}$ seconds. The overall floating point performance is now computed by dividing the number of floating point operations $N \times f$ by the time it takes to perform the computations. The overall time t_{total} is given by

$$\begin{aligned} t_{total} &= t_{reg} + t_{crs} + t_{syn} \\ &= \frac{N_r f}{P_{reg}} + \frac{(1-r)Nf}{P_{crs}} + \frac{2(1-r)Nm}{P_{reg}} \\ &= \frac{rNf}{P_{reg}} + \frac{s(1-r)Nf}{P_{reg}} + \frac{2(1-r)Nf\frac{m}{f}}{P_{reg}} \\ &= \frac{\left(r + s(1-r) + 2\frac{m}{f}(1-r)\right)Nf}{P_{reg}} \end{aligned}$$

Hence we obtain for the overall performance P_{total}:

$$\begin{aligned} P_{total} &= \frac{Nf}{t_{total}} = \frac{1}{r + s(1-r) + 2\frac{m}{f}(1-r)} P_{reg} \\ &= \frac{1}{1 + \left((s-1) + 2\frac{m}{f}\right)(1-r)} P_{reg} \end{aligned}$$

Even when ignoring the communication (m=0), the expression for P_{total} shows that with $r \to 1$, the denominator tends to one and the overall performance approaches the performance of the structured implementation. However, for a given level of refinement, it depends on the observed values for s and m whether the structured implementation is faster than the unstructured one. In the next section, we illustrate that we do achieve an acceleration.

6 Numerical Results

The stencil based representation of the discrete operator together with the block wise storage scheme for the unknown variables on the grid can be employed in all iterative solution algorithms for the equivalent linear system, that involve the application of an operator to a variable, more commonly known as matrix-vector product. We are particularly interested in geometric multigrid methods due to their optimal asymptotic complexity. Therefore we have chosen the refinement rule for our implementation in such a way that the resulting grids are nested.

Our main concern in this section is the floating point performance of the resulting program. As a consequence, we choose a test problem for which a standard multigrid algorithm is appropriate. Given that multigrid methods require computations on different levels, not only the finest one, they present a harder test case than single level Krylov methods for instance, that only involve operator applications on the finest and for our data structures most favourable level.

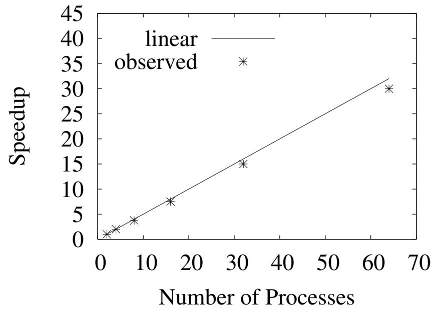
Our test case is a three dimensional Poisson problem with Dirichlet boundary conditions on L-shaped domains. The input grid is refined seven times for the scale up experiment and six times for the speed up computations. The meshsize in the different grids for the scale up experiment is constant. The domain is extended in one direction by the necessary number of equally refined unit cubes in order to fill the available processors.

The algorithmic components of the full multigrid method (FMG) are essentially standard: We employ a row-wise red-black Gauss-Seidel smoother, which updates the unknowns in a row in a red-black ordering thus breaking the data dependencies between two neighbouring points in a lexicographic numbering, and combine it with full weighting and trilinear interpolation. On each level in the multigrid hierarchy we perform two V(2,2) cycles before prolongating the obtained approximation to the next refinement level. Trilinear finite elements result in a 27 point stencil in the interior of the domain.

The results in Fig. 3, see also [6], underline that the proposed regularity oriented data structures result in a program that shows both good scale up and good speed up behaviour. Please note that the solution time for the largest problem involving more

CPU	Dof $\times 10^6$	Time in (s)
64	1179,48	44
128	2359,74	44
256	4719,47	44
512	9438,94	45
550	10139,49	48

(a)



(b)

Fig. 3. Parallel performance results in 3D: (**a**) Scalability experiments using a Poisson problem with Dirichlet boundary conditions. Each partition in the scalability experiments consists of nine cubes, each of which is regularly subdivided seven times. The timing results given refer to the wall clock time for the solution of the linear system using a full multigrid solver. (**b**) Speedup results for the same Poisson problem. The problem domain consisted of 128 cubes, each of which was subdivided six times. The problem domain was distributed to 2, 4, 8, 16, 32 and 64 processes

than 10^{10} unknowns is less than 50 seconds, on 550 processors. But this demonstrates that efficient hierarchical algorithms, in this case full multigrid, in combination with "hardware-aware" (or: *architecture-friendly*) data structures are capable of dealing with large scale problems in an acceptable amount of time. Due to the memory requirements of the matrix storage, it would not have been possible to treat a problem of that size even on the whole machine with a sparse matrix based scheme.

At the end of this section, we come back to the question of whether this implementation is actually faster than an unstructured one. Profiling results indicate that the processes in the scale up experiment spend slightly more than 30% in the Gauß-Seidel routine for the regular regions inside the cubes. The speed up factor for this routine is roughly 14 (more than 700 MFLOP/s vs. less than 50 MFLOP/s). In other words, if only this one routine was implemented in a standard, unstructured manner, the run time of the single routine would be more than four times the run time for the whole, structured program. On this architecture, our proposed approach is successful in reducing the overall solution time.

7 Conclusion

In this article, we have shown that repeatedly refined, unstructured grids, as they commonly arise in geometric multigrid methods, result in different regular features in the discretisation. We have presented data structures that capture these regularities and we have illustrated that operations on these data structures achieve a higher floating point performance than their commonly used unstructured counterparts. The impact of necessary data synchronisation on the overall performance was considered. By providing performance data from numerical experiments we have demonstrated that the proposed data structures are well suited for large scale, parallel computations with efficient individual processes.

References

1. R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd ed., SIAM, Philadelphia, 1994,
2. B. Bergen and F. Hülsemann. Hierarchical hybrid grids: A framework for efficient multigrid on high performance architectures. Technical Report 03-5, Lehrstuhl für Informatik 10, Universität Erlangen-Nürnberg, 2003
3. B. Bergen and F. Hülsemann. Hierarchical hybrid grids: data structures and core algorithms for multigrid. *Numerical Linear Algebra with Applications*, 11 (2004), pp. 279–291
4. A. Brandt. Multi–level adaptive solutions to boundary–value problems. *Math. Comp.*, 31 (1977), pp. 333–390.
5. F. Hülsemann, B. Bergen, and U. Rüde. Hierarchical hybrid grids as basis for parallel numerical solution of PDE. In H. Kosch, L. Böszörményi and H. Hellwagner (eds.) *Euro-Par 2003 Parallel Processing*, Lecture Notes in Computer Science Vol. 2790, Springer, Berlin, 2003, pp. 840–843
6. F. Hülsemann, S. Meinlschmidt, B. Bergen, G. Greiner, and U. Rüde. *gridlib* – A parallel, object-oriented framework for hierarchical-hybrid grid structures in technical simulation and scientific visualization. In *High Performance Computing in Science and Engineering*, Munich 2004. Transactions of the Second Joint HLRB and KONWIHR Result and Reviewing Workshop, Springer, Berlin, 2004, pp. 37–50
7. U. Trottenberg, C.W. Oosterlee, and A. Schüller. Multigrid. Academic Press, London, 2000

Analyzing Advanced PDE Solvers Through Simulation

Henrik Johansson, Dan Wallin, and Sverker Holmgren

Department of Information Technology, Uppsala University
Box 337, S-751 05 Uppsala, Sweden
{henrik.johansson,dan.wallin,sverker.holmgren}@it.uu.se

Abstract. By simulating a real computer it is possible to gain a detailed knowledge of the cache memory utilization of an application, e.g., a partial differential equation (PDE) solver. Using this knowledge, we can discover regions with intricate cache memory performance. Furthermore, this information makes it possible to identify performance bottlenecks.
In this paper, we employ full system simulation of a shared memory computer to perform a case study of three different PDE solver kernels with respect to cache memory performance. The kernels implement state-of-the-art solution algorithms for complex application problems and the simulations are performed for data sets of realistic size. We discovered interesting properties in the solvers, which can help us to improve their performance in the future.

1 Introduction

The performance of PDE solvers depends on many computer architecture properties. The ability to efficiently use cache memory is one such property. To analyze cache behavior, hardware counters can be used to gather rudimentary data like cache hit rate. However, such experiments can not identify the reason for a cache miss. By software simulation of a computer, it is possible to derive, e.g., the type of a cache miss and the amount of address and data traffic. A further advantage is that the simulated computer can be a fictitious system.

In this paper, we present a case study of the cache memory behavior of three PDE solvers. The solvers are based on modern, efficient algorithms, and the settings and working sets are carefully chosen to represent realistic application problems. The study is based on full-system simulations of a shared memory multiprocessor, where the baseline computer model is set up to correspond to a commercially available system, the SunFire 6800 server. However, using a simulator, the model can easily be modified to resemble alternative design choices.

We perform simulations where both the cache size and the cache line size is varied, as two simulation case studies. Many other parameters can be varied as well, but the chosen parameters show some typical properties of the studied PDE solvers.

We begin this paper with a general discussion of simulation, coupled with a description of our simulation platform (section two). Next, we present the PDE solvers used for the simulations (section three). The fourth section holds the results of the simulations, divided into two case studies. Finally, our conclusions are presented in section five.

2 Related Work

Early work in full system simulation was performed at Stanford University and resulted in the SimOS simulator [2]. The SimOS was, among others, used to study the memory usage of commericial workloads [3], data locality on CC-NUMA systems [4] and to characterize and optimize an auto-parallelizing compiler [5]. However, the work on SimOS was discontinued in 1998.

Most recent work in the field is based on the Simics full system simulator [1]. Simics has been used in a large number of publications. These include an examination of the memory system behavior for Java-based middleware [6] and a comparision of memory system behavior in java and non-java commercial workloads [7].

The work presented above is primarily concerned with hardware. In the papers real applications are simulated with the goal of improving the underlying hardware. Our paper uses the opposite approach, it uses simulation in an effort to improve the software.

3 Simulation Techniques

There are several ways to analyze algorithms on complex high performance computer systems, e.g., a high-end server. One alternative is to perform measurements on the actual physical machine, either using hardware or software. Another alternative is to develop analytical or statistical models of the system. Finally, there is simulation.

Simulation has several advantages over the other methods. First, the characteristics of the simulated computer can be set almost arbitrary. Second, the simulated machine is observable and controllable at all times. Third, a simulation is deterministic; it is always possible to replicate the results obtained from the simulation. Fourth, a high degree of automation is possible.

The ability to change parameters in the area of interest gives a researcher great flexibility. Looking at cache memory, we can easily change cache line size, cache size, replacement policy, associativity and use sub-blocked or unblocked caches. We can also determine which information to collect. In this case it can, e.g., be the number of memory accesses, cache miss rates, cache miss type and the amount of data or address traffic.

There are also a few drawbacks inherent in simulation. It is in practice impossible to build a simulator that mimics the exact behavior of the target computer in every possible situation. This might lead to some differences between the collected information and the performance of the real computer. However, comparisons between the individual simulations are valid since they all encounter the same discrepancies. Estimating execution time is difficult based on simulation and it is highly implementation dependent on the bandwidth assumptions for the memory hierarchy and the bandwidth of coherence broadcast and data switches. The complexity of an “exact” simulation is thus overwhelming, and performance and implementation issues make approximations necessary.

The experiments are carried out using execution-driven simulation in the Simics full-system simulator [1]. Simics can simulate computer architectures at the level of individual instructions. The target computer runs unmodified programs and even a complete operating system. Usually an operating system is booted inside Simics and programs, benchmarks and tests are run on this operating system in a normal fashion. Simics collects detailed statistics about the target computer and the software it runs. The user determines what information to collect, and how to do it. For example, the user can initially collect a wide range of data about the cache memory system as a whole. Later, in order to isolate a certain behavior, it is possible to restrict the collection to a small region of the code.

The modeled system implements a snoop-based invalidation MOSI cache coherence protocol [8]. We set up the baseline cache hierarchy to resemble a SunFire 6800 server with 16 processors. The server uses UltraSPARC III processors, each equipped with two levels of caches. The processors have two separate first level caches, a 4-way associative 32 KB instruction cache and a 4-way associative 64 KB data cache. The second level cache is a shared 2-way associative cache of 8 MB and the cache line size is 64 B.

4 The PDE Solvers

The kernels studied represent three types of PDE solvers, used for compressible flow computations in computational fluid dynamics (CFD), radar cross-section computations in computational electro-magnetics (CEM) and chemical reaction rate computations in quantum dynamics (QD). The properties of the kernels differ with respect to the amount of computations performed per memory access, memory access patterns and the amount of communication and its patterns.

The CFD kernel implements the advanced algorithm described by Jameson and Caughey for solving the compressible Euler equations on a 3D structured grid using a Gauss-Seidel-Newton technique combined with multi-grid acceleration [9]. The implementation is described in detail by Nordén [10]. The data is highly structured. The total work is dominated by local computations needed to update each cell. These computations are quite involved, but their parallelization is trivial. Each of the threads computes the updates for a slice of each grid in the multi-grid scheme. The amount of communication is small and the threads only communicate pair-wise.

The CEM kernel is part of an industrial solver for determining the radar cross section of an object [11]. The solver utilizes an unstructured grid in three dimensions in combination with a finite element discretization. The resulting large system of equations is solved with a version of the conjugate gradient method. In each conjugate gradient iteration, the dominating operation is a matrix-vector multiplication with the very sparse and unstructured coefficient matrix. Here, the parallelization is performed such that each thread computes a block of entries in the result vector. The effect is that the data vector is accessed in a seemingly random way. However, the memory access pattern does not change between the iterations.

The QD kernel is derived from an application where the dynamics of chemical reactions is studied using a quantum mechanical model with three degrees of freedom [12]. The solver utilizes a pseudo-spectral discretization in the two first dimensions and a finite difference scheme in the third direction. In time, an explicit ODE-solver is used. For computing the derivatives in the pseudo-spectral discretization, a standard convolution technique involving 2D fast Fourier transforms (FFTs) is applied. The parallelization is performed such that the FFTs in the x-y dimensions are performed in parallel and locally [13]. The communication within the kernel is concentrated to a transpose operation, which involves heavy all-to-all communication between the threads. However, the communication pattern is constant between the iterations in the time loop.

5 Results

We will now present two case studies performed on the PDE solvers. In the first study we simulate the solvers with different sizes of the cache memory, varied between 512 KB and 16 MB. The cache line size is held constant at 64 B. This gives a good overview of the different characteristics exhibited by the solvers. In the second study, we fix the cache memory size and vary the cache line size between 32 B and 1024 B. These simulations provide us with detailed knowledge of the cache memory utilization for each of the three solvers. The case studies should be seen as possible studies to perform in a simulated environment. Other interesting studies could for example be to vary the cache associativity or the replacement policy [14,15].

We only perform a small number of iterative steps for each PDE kernel since the access pattern is regular within each iteration. Before starting the measurements, each solver completes a full iteration to warm-up the caches. The CFD problem has a grid size of $32\times32\times32$ elements using four multi-grid levels. The CEM problem represents a modeled generic aircraft with a problem coefficient matrix of about $175{,}000\times175{,}000$ elements; a little more than 300,000 of these are non-zero. The QD problem size is $256\times256\times20$, i.e. a 256×256 2D FFT in the x-y plane followed by a FDM with 20 grid points in the z-direction. The wall clock time needed to perform the the simulations ranged from six up to twelve hours.

The cache misses are categorized into five different cache miss types according to Eggers and Jeremiassen [16]. *Cold* cache misses occur when data is accessed for the first time. A*capacity* cache miss is encountered when the cache memory is full. We also treat *conflict* misses, when data is mapped to an already occupied location in the cache, as capacity misses. When shared data is modified, it can result in a *true* sharing miss when another processor later needs the modified data. A *false* sharing cache miss results when a cache line is invalidated in order to maintain coherency, but no data was actually shared. *Updates* are not cache misses, but the messages generated in order to invalidate a shared cache line. An update can result in a sharing miss. The *data traffic* is a measurement of the number of bytes transferred on the interconnect, while the term *address traffic* represents the number of snoops the caches have to perform. All results presented are from the second level cache.

5.1 Case Study 1: Varying the Cache Size

The CFD-problem has the lowest miss rate of all the kernels. There is a relatively large amount of capacity misses when the cache size is small. This is expected, as the problem sizes for all of the applications are chosen to represent realistic work loads. There is a drop in the miss rate at a cache size of 1 MB. The drop corresponds to that all of the data used to interpolate the solution to the finest grid fits in the cache.

The CEM solver has a higher miss rate than the other applications. The miss rate decreases rapidly with larger cache sizes, especially between 2 MB and 4 MB. This indicates that the vector used in the matrix-vector multiplication now fits into the cache. The large drop in miss rate between 8 MB and 16 MB tells us that cache now is large enough to hold all the data. Some of the capacity misses become true sharing misses when the size is increased.

The QD-solver has a miss rate in-between the two other solvers. The miss rate is stable and constant until the whole problem fits into the cache. Most of the capacity misses have then been transformed into true sharing misses because of the classification scheme; the classification of capacity misses take precedence over the true sharing misses.

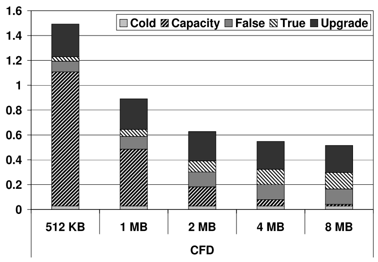
The distribution of the different miss types varies between the three solvers. All of the solvers have a lot of capacity misses when the cache size is small. The capacity misses tend to disappear when the cache size is large enough to hold the full problem. The CFD-kernel exhibits large percentage of false sharing misses. The CEM-solver is dominated by true sharing misses, while the QD-solver has equal quantities of upgrades and true sharing misses for large cache sizes. For all solvers, both data and address traffic follow the general trends seen in the miss rate.

5.2 Case Study 2: Varying the Cache Line Size

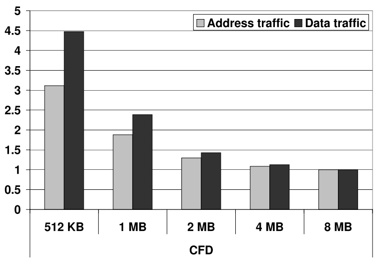
The CFD kernel performs most of the computations locally within each cell in the grid, leading to a low overall miss ratio. The data causing the true sharing misses and the upgrades exhibit good spatial locality. However, the true and false sharing misses decrease more slowly since they cannot be reduced below a certain level. The reduction in address traffic is proportional to the decrease in miss ratio. The decrease in cache miss ratio is influenced by a remarkable property: the false sharing misses are reduced when the line size is increased. The behavior of false sharing is normally the opposite; false sharing misses increase with larger line size. Data shared over processor boundaries are always, but incorrectly, invalidated. If a shorter cache line is used, less invalidated data will be brought into the local cache after a cache miss and accordingly, a larger number of accesses is required to bring all the requested data to the cache [14]. It is probable that this property can be exploited to improve the cache memory utilization.

The CEM kernel has a large problem size, causing a high miss ratio for small cache line sizes. Capacity misses are common and can be avoided using a large cache line size. The true sharing misses and the upgrades are also reduced with a larger cache line size, but at a slower rate since the data vector is being irregular accessed. False sharing is never a problem in this application. Both the data and address traffic exhibit nice properties for large cache lines because of the rapid decrease in the miss ratio with increased line size.

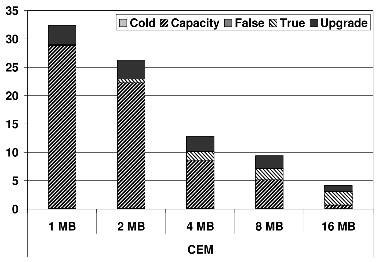
The QD kernel is heavily dominated by two miss types, capacity misses and upgrades, which both decrease with enlarged cache line size. The large number of upgrades is caused by the all-to-all communication pattern during the transpose operation where every element is modified after being read. This should result in an equal number of true sharing misses and upgrades, but the kernel problem size is very large and replacements take place before the true sharing misses occur. Therefore, a roughly equal amount of capacity misses and upgrades are recorded. The data traffic increases rather slowly for cache lines below 512 B. The address traffic shows the opposite behavior and decreases rapidly until the 256 B cache line size is reached, where it levels out.



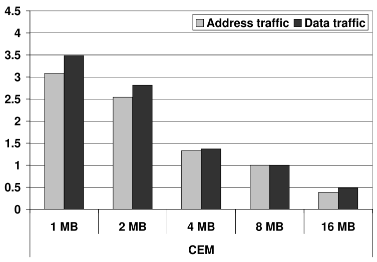
(a) CFDm



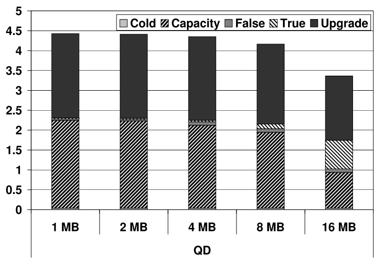
(b) CFDt



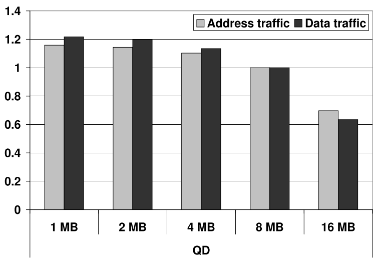
(c) CEMm



(d) CEMt



(e) QDm



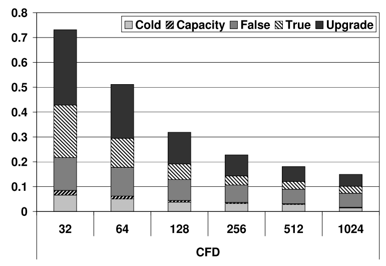
(f) QDt

Fig. 1. Influence of cache size on cache misses, address and data traffic. The miss ratio in percent is indicated in the cache miss figures. The address and data traffic are normalized relative to the 8 MB configuration

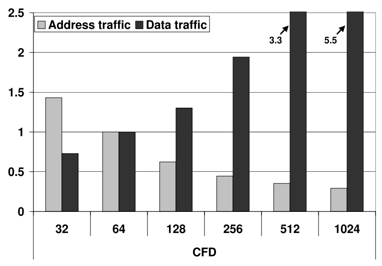
6 Conclusion

We have shown that full system simulation of PDE solvers with realistic problem sizes is possible. By using simulation, we can obtain important information about the run time behavior of the solvers. The information is difficult to collect using traditional methods such as hardware counters. Simulation also allows us to run the solvers on non existing computer systems and configurations.

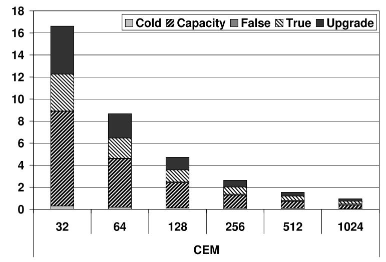
Our three PDE-solvers were found to have vastly different cache memory characteristics. The collected data in combination with a good knowledge of the algorithms made it possible to understand the behavior of each solver in great detail. We are certain that this Understanding will make it easier to find ways to better utilize the cache memory.



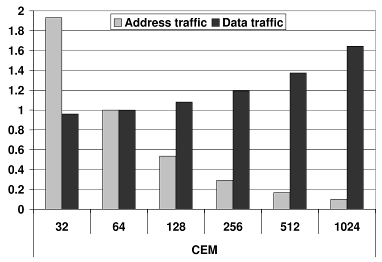
(a) CFDm



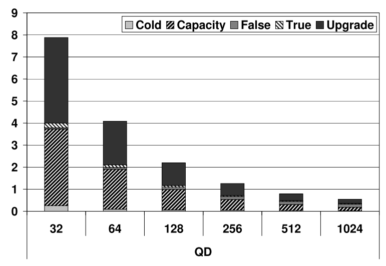
(b) CFDt



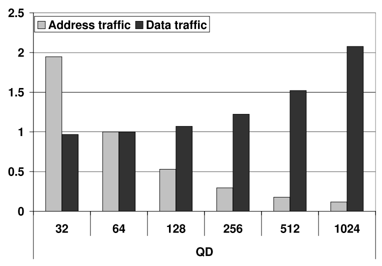
(c) CEMm



(d) CEMt



(e) QDm



(f) QDt

Fig. 2. Influence of cache line size on cache misses, address and data traffic. The miss ratio in percent is indicated in the cache miss figures. The address and data traffic are normalized relative to the 64 B configuration

References

1. P. S. Magnusson et al, Simics: A Full System Simulation Platform, IEEE Computer, Vol. 35, No. 2, pp. 50-58, 2002.
2. M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod,Using the SimOS Machine Simulator to Study Complex Computer Systems, ACM TOMACS Special Issue on Computer Simulation,1997.
3. L. A. Barroso, K. Gharachorloo and E. Bugnion, Memory System Characterization of Commercial Workloads, In Proceedings of the 25th International Symposium on Computer Architecture, June 1998.
4. B. Verghese, S. Devine, A. Gupta, and M. Rosenblum, Operating System Support for Improving Data Locality on CC-NUMA Compute servers, In ASPLOS VII, Cambridge, MA, 1996.

5. E. Bugnion, J. M. Anderson and M. Rosenblum, Using SimOS to characterize and optimize auto-parallelized SUIF applications, First SUIF Compiler Workshop, Stanford University, Jan. 11-13, 1996.
6. M. Karlsson, K. Moore, E. Hagersten, and D. Wood,Memory System Behavior of Java-Based Middleware, n Proceedings of the Ninth International Symposium on High Performance Computer Architecture (HPCA-9), Anaheim, California, USA, February 2003.
7. M. Marden, S. Lu, K. Lai, M. Lipasti, Comparison of Memory System Behavior in Java and Non-Java Commercial Workloads, In the proceedings of the Computer Architecture Evaluation using Commercial Workloads (CAECW-02), February 2, 2002.
8. J. Hennessy and D Patterson, Computer Architechture, A Quantitative Approach, Morgan Kaufmann.
9. A. Jameson and D. A. Caughey, How Many Steps are Required to Solve the Euler Equations of Steady Compressible Flow: In Search of a Fast Solution Algorithm, AIAA 2001-2673, 15th AIAA Computational Fluid Dynamics Conference, June 11-14, 2001, Anaheim, CA.
10. M. Nordén, M. Silva, S. Holmgren, M. Thuné and R. Wait, Implementation Issues for High Performance CFD, Proceedings of International Information Technology Conference, Colombo, 2002.
11. F. Edelvik, Hybrid Solvers for the Maxwell Equations in Time-Domain, PhD thesis, Dep. of Information Technology, Uppsala University, 2002.
12. Å. Petersson, H. Karlsson and S. Holmgren, Predissociation of the Ar-12 van der Waals Molecule, a 3D Study Performed Using Parallel Computers, Submitted to Journal of Physical Chemistry, 2002.
13. D. Wallin, Performance of a High-Accuracy PDE Solver on a Self-Optimizing NUMA Architecture, Master's thesis, Dep. of Information Technology, Uppsala University, 2001.
14. H. Johansson, An Analysis of Three Different PDE-Solvers With Respect to Cache Performance, Master's thesis, Dep. of Information Technology, Uppsala University, 2003.
15. D. Wallin, H. Johansson and S. Holmgren, Cache Memory Behavior of Advanced PDE Solvers, Parallel Computing: Software Technology, Algorithms, Architectures and Applications 13, Proceedings of the International Conference ParCo2003, pp. 475-482,Dresden, Germany, 2003.
16. S. J. Eggers and T. E. Jeremiassen, Eliminating False Sharing, Proceedings of International Conference on Parallel Processing, pp. 377-381, 1991.

Towards Cache-Optimized Multigrid Using Patch-Adaptive Relaxation

Markus Kowarschik, Iris Christadler, and Ulrich Rüde

System Simulation Group, Computer Science Department
Friedrich-Alexander-University Erlangen-Nuremberg, Germany
{Markus.Kowarschik,Iris.Christadler,Ulrich.Ruede}@cs.fau.de

Abstract. Most of today's computer architectures employ fast, yet relatively small cache memories in order to mitigate the effects of the constantly widening gap between CPU speed and main memory performance. Efficient execution of numerically intensive programs can only be expected if these hierarchical memory designs are respected. Our work targets the optimization of the cache performance of multigrid codes. The research efforts we will present in this paper first cover transformations that may be automized and then focus on fundamental algorithmic modifications which require careful mathematical analysis. We will present experimental results for the latter.

1 Introduction

In order to mitigate the effects of the constantly widening gap between CPU speed and main memory performance, today's computer architectures typically employ hierarchical memory designs involving several layers of fast, yet rather small cache memories (caches) [6]. Caches contain copies of frequently used data.

However, efficient code execution can only be expected if the codes respect the structure of the underlying memory subsystem. Unfortunately, current optimizing compilers are not able to synthesize chains of complicated cache-based code transformations. Hence, they rarely deliver the performance expected by the users and much of the tedious and error-prone work concerning the tuning of the memory efficiency is thus left to the software developer [3,8].

The research we will present in this paper focuses on the optimization of the data cache performance of multigrid methods which have been shown to be among the fastest numerical solution methods for large sparse linear systems arising from the discretization of elliptic partial differential equations (PDEs) [14]. In particular, we will concentrate on the optimization of the smoother which is typically the most time-consuming part of a multigrid implementation [7,15]. The smoother is employed in order to eliminate the highly oscillating components of the error on the corresponding grid levels. Conversely, the coarse-grid correction is responsible for the reduction of the slowly oscillating error frequencies.

Our techniques can be divided into two categories. On the one hand, we will briefly address "compiler-oriented" cache optimization approaches which do not modify the results of the numerical computations and may therefore be fully automized using a compiler or a source code restructuring tool; e.g., ROSE [11]. On the other hand, we will describe a more fundamental approach which is derived from the theory of the *fully adaptive multigrid method* [13] and based on a patch-adaptive processing strategy. This novel algorithm requires careful mathematical analysis and is therefore a long way from being automatically introduced by a compiler [7].

See [3,8,9] for comprehensive surveys on memory hierarchy optimizations. In particular, cache performance optimizations for computations on structured grids have further been presented in [7,12,15]. For the case of unstructured grids, we particularly refer to [2].

Our paper is structured as follows. Compiler-oriented cache-based transformations will be reviewed briefly in Section 2. Section 3 contains the description of the adaptive relaxation schemes. Afterwards, experimental results will be demonstrated in Section 4. In Section 5, we will draw some final conclusions.

2 Compiler-Oriented Data Locality Optimizations

2.1 Data Layout Optimizations

Data layout optimizations aim at enhancing code performance by improving the arrangement of the data in address space [7]. On the one hand, such techniques are applied to change the mapping of array data to the cache frames, thereby reducing the number of cache conflict misses and avoiding cache thrashing [3]. This is commonly achieved by a layout transformation called *array padding* which introduces additional array elements that are never referenced in order to modify the relative distances of array entries in address space [12].

On the other hand, data layout optimizations can be applied to increase spatial locality. In particular, they can be introduced in order to enhance the reuse of cache blocks once they have been loaded into cache. Since cache blocks contain several data items that are arranged next to each other in address space it is reasonable to aggregate those data items which are likely to be referenced within a short period of time. This technique is called *array merging* [6].

For a detailed discussion of data layout optimizations for multigrid codes, we again refer to [7,15].

2.2 Data Access Optimizations

Data access optimizations are code transformations which change the order in which iterations in a loop nest are executed. These transformations strive to improve both spatial and temporal locality. Moreover, they can also expose parallelism and make loop iterations vectorizable.

Classical loop transformations cover *loop fusion* as well as *loop blocking (tiling)*, amongst others [1]. Loop fusion refers to a transformation which takes two adjacent loops that have the same iteration space traversal and combines their bodies into a single loop, thereby enhancing temporal locality. Furthermore, fusing two loops results in a single loop which contains more instructions in its body and thus offers increased instruction-level parallelism. Finally, only one loop is executed, thus reducing the total loop overhead by a factor of 0.5 [8].

Loop blocking refers to a loop transformation which increases the depth of a loop nest with depth d by adding additional loops. The depth of the resulting loop nest will be anything from $d+1$ to $2d$. Loop blocking is primarily used to improve temporal locality by enhancing the reuse of data in cache and reducing the number of cache capacity misses. Tiling a single loop replaces it by a pair of loops. The inner loop of the new loop nest traverses a *block (tile)* of the original iteration space with the same increment as the original loop. The outer loop traverses the original iteration space with an increment equal to the size of the block which is traversed by the inner loop. Thus, the outer loop feeds blocks of the whole iteration space to the inner loop which then executes them step by step [8].

As is the case for data layout transformations, a comprehensive overview of data access optimizations for multigrid codes can be found in [7,15]. See [2] for blocking approaches for unstructured grids.

3 Enhancing Data Locality Through Adaptive Relaxation

3.1 Overview: Fully Adaptive Multigrid

Our novel patch-adaptive multigrid scheme is derived from the theory of the fully adaptive multigrid method. The latter combines two different kinds of *adaptivity* into a single algorithmic framework [13]. Firstly, *adaptive mesh refinement* is taken into account in order to reduce the number of degrees of freedom and, as a consequence, the computational work as well as the memory consumption of the implementation. Secondly, the fully adaptive multigrid method employs the principle of *adaptive relaxation* [13]. With this update strategy, computational work can be focused on where it can efficiently improve the quality of the numerical approximation.

In this paper, we will concentrate on the principle of adaptive relaxation only and demonstrate how it can be exploited to enhance cache efficiency. For the sake of simplicity, we will restrict the following presentation of the algorithms to a single grid level and thus omit all level indices. However, due to the uniform reduction of the algebraic error, this scheme reveals its full potential when used as a smoother in a multilevel structure [13].

3.2 Preliminaries

We consider the system

$$Ax = b \ , \ \ A = (a_{i,j})_{1 \le i,j \le n} \ , \tag{3.1}$$

of linear equations and assume that the matrix A is sparse and symmetric positive definite. As a common example, standard second-order finite difference discretizations of the negative Laplacian yield such matrices. The exact solution of (3.1) shall be denoted as x^*.

We assume that x stands for an approximation to x^*. The *scaled residual* $\bar{r}$ corresponding to x is then defined as $\bar{r} := D^{-1}(b - Ax)$, where D denotes the diagonal part of A. With e_i representing the i-th unit vector, the components θ_i, of the scaled residual $\bar{r}$ are given by $\theta_i := e_i^T \bar{r}$, $1 \le i \le n$.

An *elementary relaxation step* for equation i of the linear system (3.1), which updates the approximation x and yields the new approximation x', is given by $x' = x + \theta_i e_i$, and the Gauss-Seidel update scheme [5] can be written as follows:

$$x_i^{(k+1)} = a_{i,i}^{-1} \left(b_i - \sum_{j<i} a_{i,j} x_j^{(k+1)} - \sum_{j>i} a_{i,j} x_j^{(k)} \right) = x_i^{(k)} + \theta_i \ .$$

Here, θ_i represents the current scaled residual of equation i; i.e., the scaled residual of equation i *immediately* before this elementary update operation. As a consequence of this elementary relaxation step, the i-th component of $\bar{r}$ vanishes.

The motivation of the development of the adaptive relaxation method is based on the following relation, which states how the algebraic error $x^* - x$ is reduced by performing a single elementary relaxation step:

$$||x^* - x||_A^2 - ||x^* - x'||_A^2 = a_{i,i}\theta_i^2 \ , \tag{3.2}$$

see [13]. As usual, $||v||_A$ stands for the energy norm of v.

Equation (3.2) implies that, with every elementary relaxation step, the approximation is either improved (if $\theta_i \neq 0$) or remains unchanged (if $\theta_i = 0$). Note that, since A is positive definite, all diagonal entries $a_{i,i}$ of A are positive. Hence, it is the positive definiteness of A which guarantees that the solution x will never become "worse" by applying elementary relaxation operations.

Equation (3.2) further represents the mathematical justification for the selection of the (numerically) most efficient equation update order. It states that the fastest error reduction is obtained by relaxing those equations i whose scaled residuals θ_i have relatively large absolute values. This is particularly exploited in the *method of Gauss-Southwell* where always the equation i with the largest value $|a_{i,i}\theta_i|$ is selected for the next elementary relaxation step. Yet, if no suitable update strategy is used, this algorithm will be rather inefficient since determining the equation whose residual has the largest absolute value is as expensive as relaxing each equation once [13].

This observation motivates the principle of the adaptive relaxation method which is based on an *active set* of equation indices and can be interpreted as an algorithmically efficient approximation to the method of Gauss-Southwell. For a concise description of the operations on the active set and the method of adaptive relaxation, it is convenient to introduce the *set of connections* of grid node i, $1 \leq i \leq n$, as

$$\mathrm{Conn}(i) := \{j; 1 \leq j \leq n,\ j \neq i,\ a_{j,i} \neq 0\} \ .$$

Intuitively, the set $\mathrm{Conn}(i)$ contains the indices of those unknowns u_j that directly depend on u_i, except for u_i itself. Note that, due to the symmetry of A, these are exactly the unknowns u_j, $j \neq i$, that u_i directly depends on.

Before we can provide an algorithmic formulation of the adaptive relaxation method, it is further necessary to introduce the *strictly active set* $S(\theta, x)$ of equation indices:

$$S(\theta, x) := \{i;\ 1 \leq i \leq n,\ |\theta_i| > \theta\} \ .$$

Hence, the set $S(\theta, x)$ contains the indices i of all equations whose current scaled residuals θ_i have absolute values greater than the prescribed threshold $\theta > 0$. Possible alternatives of this definition are discussed in [13].

Finally, an *active set* $\tilde{S}(\theta, x)$ is a superset of the strictly active set $S(\theta, x)$:

$$S(\theta, x) \subseteq \tilde{S}(\theta, x) \subseteq \{i;\ 1 \leq i \leq n\} \ .$$

Note that $\tilde{S}(\theta, x)$ may also contain indices of equations whose scaled residuals have relatively small absolute values. The idea behind the introduction of such an extended active set is that it can be maintained efficiently. This is exploited by the adaptive relaxation algorithms that we will present next.

3.3 Point-Based Adaptive Relaxation

The core of the *sequential (Gauss-Seidel-type) adaptive relaxation method*[1] is presented in Algorithm 1. Note that it is essential to initialize the active set $\tilde{S}$ appropriately. Unless problem-specific information is known a priori, $\tilde{S}$ is initialized with all indices i, $1 \leq i \leq n$.

Algorithm 1 Sequential adaptive relaxation

```
Require: tolerance $\theta > 0$, initial guess $x$, initial active set $\tilde{S} \subseteq \{i;\ 1 \leq i \leq n\}$
1: while $\tilde{S} \neq \emptyset$ do
2:    Pick $i \in \tilde{S}$ {Nondeterministic choice}
3:    $\tilde{S} \leftarrow \tilde{S} \setminus \{i\}$
4:    if $|\theta_i| > \theta$ then
5:       $x \leftarrow x + \theta_i e_i$ {Elementary relaxation step}
6:       $\tilde{S} \leftarrow \tilde{S} \cup \mathrm{Conn}(i)$
7:    end if
8: end while
Ensure: $S = \emptyset$ {The strictly active set $S$ is empty}
```

The algorithm proceeds as follows. Elementary relaxation steps are performed until the active set $\tilde{S}$ is empty, which implies that none of the absolute values of the scaled residuals θ_i exceeds the given tolerance θ anymore. In Steps 2 and 3, an equation index $i \in \tilde{S}$ is selected and removed from $\tilde{S}$. Only if the absolute value of the corresponding scaled residual θ_i is larger than θ (Step 4), equation i will be relaxed. The relaxation of equation i implies that all scaled residuals θ_j, $j \in \mathrm{Conn}(i)$, are changed, and it must therefore be checked subsequently if their absolute values $|\theta_j|$ now exceed the tolerance θ. Hence, if equation i is relaxed (Step 5), the indices $j \in \mathrm{Conn}(i)$ will be put into the active set $\tilde{S}$ (Step 6), unless they are already contained in $\tilde{S}$.

As was mentioned above, the full advantage of the adaptive relaxation scheme becomes evident as soon as it is introduced into a multigrid structure. The idea is to employ this method as a smoother in a multigrid setting, where it is not necessary to completely eliminate the errors on the finer grids. Instead, it is enough to smooth the algebraic errors such that reasonably accurate corrections can be computed efficiently on the respective coarser grids [14].

[1] A *simultaneous (Jacobi-type)* adaptive relaxation scheme is also discussed in [13].

The application of the adaptive relaxation method in the multigrid context corresponds to the approach of *local relaxation*. The principle of local relaxation states that the robustness of multigrid algorithms can be improved significantly by performing additional relaxation steps in the vicinity of singularities or perturbations from boundary conditions, for example. The purpose of these additional elementary relaxation steps is to enforce a sufficiently smooth error such that the subsequent coarse-grid correction performs well. We refer to [13] for further details and especially to the references provided therein.

At this point, we eventually have everything in place to introduce our novel patch-adaptive relaxation scheme which can be considered cache-aware by construction.

3.4 Cache-Optimized Patch-Based Adaptive Relaxation

A *patch* can be interpreted as a frame (window) that specifies a region of a computational grid. For ease of presentation, we only consider non-overlapping patches. Our *sequential patch-based adaptive relaxation scheme* parallels the point-oriented version from Section 3.3. This implies that, when introduced into a multigrid structure, it also follows the aforementioned principle of local relaxation and, in general, behaves in a more robust fashion than standard multigrid algorithms.

In the case of patch-adaptive relaxation, however, the active set $\tilde{S}$ does not contain the indices of individual equations (i.e., the indices of individual grid nodes). Instead, $\tilde{S}$ contains the indices of patches. The set of all patches is denoted as $\mathcal{P}$. Since the number of patches is usually much smaller than the number of unknowns, the granularity of the adaptive relaxation scheme and, in addition, the overhead of maintaining the active set during the iteration are both reduced significantly.

Furthermore, the patch-adaptive relaxation is characterized by its inherent data locality. Consequently, it leads to a high utilization of the cache and therefore to fast code execution. This is accomplished through the use of appropriately sized patches as well as through the repeated relaxation of a patch once it has been loaded into cache. It has been demonstrated that, once an appropriately sized patch has been loaded into the cache and updated for the first time, the costs of subsequently relaxing it a few more times are relatively low [10]. See [7] for details.

The core of the sequential patch-adaptive relaxation scheme is presented in Algorithm 2[2]. Upon termination, the strictly active set S is empty and, therefore, each scaled residual θ_i is less or equal than the prescribed threshold θ, $1 \leq i \leq n$. In Step 4, $\bar{r}_P$ is used to represent the scaled residual corresponding to patch P (i.e., to the nodes covered by P) and $||\bar{r}_P||_\infty$ denotes its maximum norm.

Relaxing a patch (Step 5) means performing Gauss-Seidel steps on its nodes until each of its own scaled residuals θ_i has fallen below the (reduced) tolerance $\theta - \delta$. As a consequence, after a patch P has been relaxed any of its neighbor patches P' will only be inserted into the active set if the absolute values of the scaled residuals of P' have been increased "significantly". This allows for the application of sophisticated activation strategies in Step 6, which aim at avoiding unnecessary patch relaxations. A

[2] As is the case for the point-based method, a simultaneous version of the patch-based scheme is also conceivable.

Algorithm 2 Sequential patch-adaptive relaxation

```
Require: tolerance $\theta > 0$, tolerance decrement $\delta = \delta(\theta)$, $0 < \delta < \theta$, initial guess $x$,
    initial active set $\tilde{S} \subseteq \mathcal{P}$
1: while $\tilde{S} \neq \emptyset$ do
2:     Pick $P \in \tilde{S}$ {Nondeterministic choice}
3:     $\tilde{S} \leftarrow \tilde{S} \setminus \{P\}$
4:     if $||\bar{r}_P||_\infty > \theta - \delta$ then
5:         relax($P$)
6:         activateNeighbors($P$)
7:     end if
8: end while
Ensure: $S = \emptyset$ {The strictly active set $S$ is empty}
```

comprehensive discussion of such strategies is beyond the scope of this paper. Instead, we again point to [7].

4 Experimental Results

Performance results that demonstrate the effectiveness of our data layout transformations as well as our data access transformations have extensively been studied in [15] for the 2D case and in [7] for the 3D case. Therefore, this section will concentrate on the efficiency of the patch-adaptive multigrid scheme.

4.1 Model Problem

We define an L-shaped domain $\Omega := \{(x, y) \in (-0.5, 0.5)^2;\ x < 0 \text{ or } y > 0\} \subset \mathbf{R}^2$ and introduce 2D polar coordinates (r, φ), $r \geq 0$, $0 \leq \varphi \leq 2\pi$, as usual; i.e., we have $x = r\cos\varphi$ and $y = r\sin\varphi$. We consider the Dirichlet boundary value problem

$$-\Delta u = 0 \quad \text{in} \quad \Omega \ , \tag{4.3}$$

$$u(r, \varphi) = \sin\left(\frac{2}{3}\varphi\right) r^{\frac{2}{3}} \quad \text{on} \quad \partial\Omega \ . \tag{4.4}$$

It is straightforward to verify that the analytical solution of (4.3), (4.4) is given by $u(r, \varphi) := \sin(\frac{2}{3}\varphi)r^{2/3}$ and that we have $u \in C^2(\Omega) \cap C^0(\bar{\Omega})$, but $u \notin C^1(\bar{\Omega})$. This singular behavior of u results from the fact that the first derivatives of u are not bounded in $r = 0$, cf. [4].

Figure 1 shows a plot of u using a grid with a mesh width of 64^{-1}. In the following, we will demonstrate how this model problem can be solved efficiently using our patch-adaptive multigrid scheme.

4.2 Numerical Tests

We have employed a hierarchy of nine regular grids with a finest grid of mesh width of 1024^{-1}. Standard coarsening has been used. The Laplacian has been discretized anew

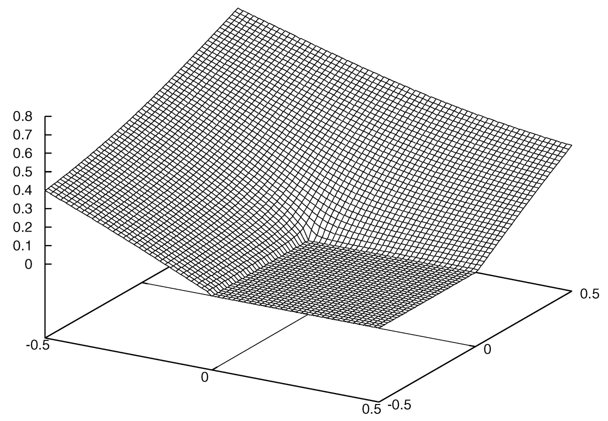


Fig. 1. Analytical solution of the model problem

on each level using bilinear basis functions on the quadrilateral finite elements. This discretization approach leads to constant 9-pont stencils on each grid level.

We have determined the total numbers of elementary relaxations steps per grid level for a multigrid V(0,4)-cycle involving a red-black Gauss-Seidel (RBGS) smoother. Additionally, we have determined the corresponding numbers for the patch-adaptive multigrid scheme. The patch-adaptive relaxation scheme from Section 3.4 has been applied to smooth the errors on the five finest levels, using a patch size of approximately 32×32 nodes. On the four coarser levels, a standard red-black Gauss-Seidel smoother has been employed. This is because it is only reasonable to apply the patch-adaptive scheme if the number of unknowns on the corresponding level is large enough such that it pays off to implement a smoother based on an active set strategy.

The results of the comparison are summarized in Table 2. They clearly emphasize the numerical efficiency of the multigrid method involving our patch-adaptive smoother. It is obvious that the patch-adaptive multigrid algorithm requires far less computational work than the standard one, even when counting all necessary residual calculations in addition to the actual number of elementary relaxation steps, cf. again Algorithm 2 from Section 3.4. This is due to the fact that numerical schemes based on adaptive relaxation generally represent good candidates for solving such problems that exhibit singularities [13]. Their applicability to "smoother" problems, however, is subject to future research.

Moreover, the RBGS-based multigrid method exhibits an average *residual reduction factor* of 0.22 per V-cycle[3]. Therefore, we have prescribed the tolerance $\theta := 0.22$ for the patch-adaptive smoother on each respective grid level, including the finest one which is typically considered solely for convergence rate measurements. The tolerance decrement δ has been chosen empirically as $\delta := 0.2\theta$. The patch-adaptive multigrid scheme has actually exhibited a significantly better residual reduction factor of 0.063. This observation again emphasizes the enhanced numerical efficiency of the patch-adaptive scheme, cf. Section 3.3.

[3] This means that we have divided the maximum norms of the scaled residuals on the finest grid level before and after each V-cycle.

		RBGS-based MG	Patch-adapt. MG	Patch-adapt. MG
				♯residual comp.
Level	♯unknowns	♯elem. rel. steps	♯elem. rel. steps	+♯elem. rel. steps
0	5	20	20	20
1	33	132	132	132
2	161	644	644	644
3	705	2 820	2 820	2 820
4	2 945	11 780	54 932	57 877
5	12 033	48 132	44 672	56 705
6	48 641	194 564	44 672	93 313
7	195 585	782 340	48 640	244 225
8	784 385	3 137 540	56 832	841 217
Total	1 044 493	4 177 972	253 364	1 296 953

Fig. 2. Comparison of standard and patch-adaptive multigrid methods

A detailed analysis of the behavior of the patch-adaptive multigrid scheme reveals that, on the coarsest level on which the patch-adaptive smoother is used, more elementary relaxation steps are performed than in the case of the standard smoother. However, this additional work pays off on the finer levels, where only those patches in the vicinity of the singularity at $r = 0$ are relaxed more intensively. We again refer to [7] for a comprehensive discussion.

Preliminary performance experiments have indicated the improved cache efficiency of the patch-adaptive relaxation scheme. However, the development of highly tuned implementations which exhibit high floating-point performance (in terms of MFLOPS rates) is still in progress.

5 Conclusions

The continued improvements in processor performance are generally placing increasing pressure on the memory hierarchy. Therefore, we have developed and examined optimization techniques in order to enhance the data locality exhibited by grid-based numerical computations, particularly the time-consuming smoothers in multigrid algorithms.

We have presented a promising approach towards the design of novel, inherently cache-aware algorithms and, so far, we have demonstrated its enhanced numerical robustness. The development of highly tuned hardware-aware implementations, however, is the focus of ongoing and future work.

References

1. R. Allen and K. Kennedy. *Optimizing Compilers for Modern Architectures*. Morgan Kaufmann, 2001.

2. C.C. Douglas, J. Hu, M. Kowarschik, U. Rüde, and C. Weiß. Cache Optimization for Structured and Unstructured Grid Multigrid. *Electronic Transactions on Numerical Analysis*, 10:21–40, 2000.
3. S. Goedecker and A. Hoisie. *Performance Optimization of Numerically Intensive Codes*. SIAM, 2001.
4. W. Hackbusch. *Elliptic Differential Equations — Theory and Numerical Treatment*, volume 18 of *Springer Series in Computational Mathematics*. Springer, 1992.
5. W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*, volume 95 of *Applied Mathematical Sciences*. Springer, 1993.
6. J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 3. edition, 2003.
7. M. Kowarschik. *Data Locality Optimizations for Iterative Numerical Algorithms and Cellular Automata on Hierarchical Memory Architectures*. PhD thesis, Lehrstuhl für Informatik 10 (Systemsimulation), Institut für Informatik, Universität Erlangen-Nürnberg, Erlangen, Germany, July 2004. SCS Publishing House.
8. M. Kowarschik and C. Weiß. An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms. In U. Meyer, P. Sanders, and J. Sibeyn, editors, *Algorithms for Memory Hierarchies — Advanced Lectures*, volume 2625 of *Lecture Notes in Computer Science*, pages 213–232. Springer, 2003.
9. D. Loshin. *Efficient Memory Programming*. McGraw-Hill, 1998.
10. H. Lötzbeyer and U. Rüde. Patch-Adaptive Multilevel Iteration. *BIT*, 37(3):739–758, 1997.
11. D. Quinlan, M. Schordan, B. Miller, and M. Kowarschik. Parallel Object-Oriented Framework Optimization. *Concurrency and Computation: Practice and Experience*, 16(2–3):293–302, 2004. Special Issue: Compilers for Parallel Computers.
12. G. Rivera and C.-W. Tseng. Tiling Optimizations for 3D Scientific Computations. In *Proc. of the ACM/IEEE Supercomputing Conf.*, Dallas, Texas, USA, 2000.
13. U. Rüde. *Mathematical and Computational Techniques for Multilevel Adaptive Methods*, volume 13 of *Frontiers in Applied Mathematics*. SIAM, 1993.
14. U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001.
15. C. Weiß. *Data Locality Optimizations for Multigrid Methods on Structured Grids*. PhD thesis, Lehrstuhl für Rechnertechnik und Rechnerorganisation, Institut für Informatik, Technische Universität München, Munich, Germany, December 2001.

Hierarchical Partitioning and Dynamic Load Balancing for Scientific Computation

James D. Teresco[1], Jamal Faik[2], and Joseph E. Flaherty[2]

[1] Department of Computer Science, Williams College
Williamstown, MA 01267 USA
terescoj@cs.williams.edu

[2] Department of Computer Science, Rensselaer Polytechnic Institute
Troy, NY 12180, USA
{faikj,flaherje}@cs.rpi.edu

Abstract. Cluster and grid computing has made hierarchical and heterogeneous computing systems increasingly common as target environments for large-scale scientific computation. A cluster may consist of a network of multiprocessors. A grid computation may involve communication across slow interfaces. Modern supercomputers are often large clusters with hierarchical network structures. For maximum efficiency, software must adapt to the computing environment. We focus on partitioning and dynamic load balancing, in particular on hierarchical procedures implemented within the Zoltan Toolkit, guided by DRUM, the Dynamic Resource Utilization Model. Here, different balancing procedures are used in different parts of the domain. Preliminary results show that hierarchical partitionings are competitive with the best traditional methods on a small hierarchical cluster.

Introduction

Modern three dimensional scientific computations must execute in parallel to achieve acceptable performance. Target parallel environments range from clusters of workstations to the largest tightly-coupled supercomputers. Hierarchical and heterogeneous systems are increasingly common as symmetric multiprocessing (SMP) nodes are combined to form the relatively small clusters found in many institutions as well as many of today's most powerful supercomputers. Network hierarchies arise as grid technologies make Internet execution more likely and modern supercomputers are built using hierarchical interconnection networks. MPI implementations may exhibit very different performance characteristics depending on the underlying network and message passing implementation (*e.g.*, [32]). Software efficiency may be improved using optimizations based on system characteristics and domain knowledge. Some have accounted for clusters of SMPs by using a hybrid programming model, with message passing for inter-node communication and multithreading for intra-node communication (*e.g.*, [1,27]), with varying degress of success, but always with an increased burden on programmers to program both levels of parallelization.

Our focus has been on resource-aware partitioning and dynamic load balancing, achieved by adjusting target partition sizes or the choice of a dynamic load-balancing procedure or its parameters, or by using a combination of load-balancing procedures. We retain the flat message passing programming model. For hierarchical and heterogeneous systems, different choices of load balancing procedures may be appropriate in different parts of the parallel environment. There are tradeoffs in execution time and partition quality (*e.g.*, surface indices, interprocess connectivity, strictness of load balance) [35] and some may be more important than others in some circumstances. For example, consider a cluster of SMP nodes connected by Ethernet. A more costly graph partitioning can be done to partition among the nodes, to minimize communication across the slow network interface, possibly at the expense of some computational imbalance. Then, a fast geometric algorithm can be used to partition independently within each node.

1 Partitioning and Dynamic Load Balancing

An effective partitioning or dynamic load balancing procedure maximizes efficiency by minimizing processor idle time and interprocessor communication. While some applications can use a static partitioning throughout a computation, others, such as adaptive finite element methods, have dynamic workloads that necessitate dynamic load balancing during the computation. Partitioning and dynamic load balancing can be performed using recursive bisection methods [2,29,31,38], space-filling curve (SFC) partitioning [7,23,24,25,37] and graph partitioning (including spectral [26,29], multilevel [6,18,20,36], and diffusive methods [8,19,22]). Each algorithm has characteristics and requirements that make it appropriate for certain applications; see [4,35] for examples and [33] for an overview of available methods.

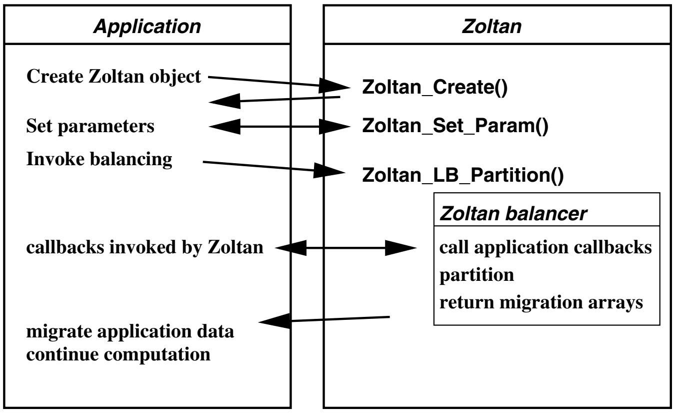


Fig. 1. Interaction between Zoltan and applications

The Zoltan Parallel Data Services Toolkit [9,11] provides dynamic load balancing and related capabilities to a wide range of dynamic, unstructured and/or adaptive applications. Using Zoltan, application developers can switch partitioners simply by changing a run-time parameter, facilitating comparisons of the partitioners' effect on the applications. Zoltan has a simple interface, and its design is "data-structure neutral." That is, Zoltan does not require applications to construct or use specific data structures. It operates on generic "objects" that are specified by calls to application-provided callback functions. These callbacks are simple functions that return to Zoltan information such as the lists of objects to be partitioned, coordinates of objects, and topological connectivity of objects. Figure 1 illustrates the interaction between Zoltan and an application.

We focus here on the hierarchical balancing procedures we have implemented within Zoltan, where different procedures are used in different parts of the computing environment.

2 Hierarchical Balancing

Consider four different 16-way partitionings of a 1,103,018-element mesh used in a simulation of blood flow in a human aorta [30]. Here, the geometric method recursive inertial bisection (RIB) [28] and/or the multilevel graph partitioner in ParMetis [20] are used for partitioning. Only two partition quality factors are considered: computational balance, and two *surface index* measures, although other factors [5] should be considered. The *global surface index* (GSI) measures the overall percentage of mesh faces inter-partition boundaries, and the *maximum local surface index* ($MLSI$) measures the maximum percentage of any one partition's faces that are on a partition boundary. The surface indices are essentially normalized edge cuts when considering mesh connectivity in terms of a graph. Partitioning using RIB achieves an excellent computational balance, with partitions differing by no more than one element, and medium-quality surface indices with

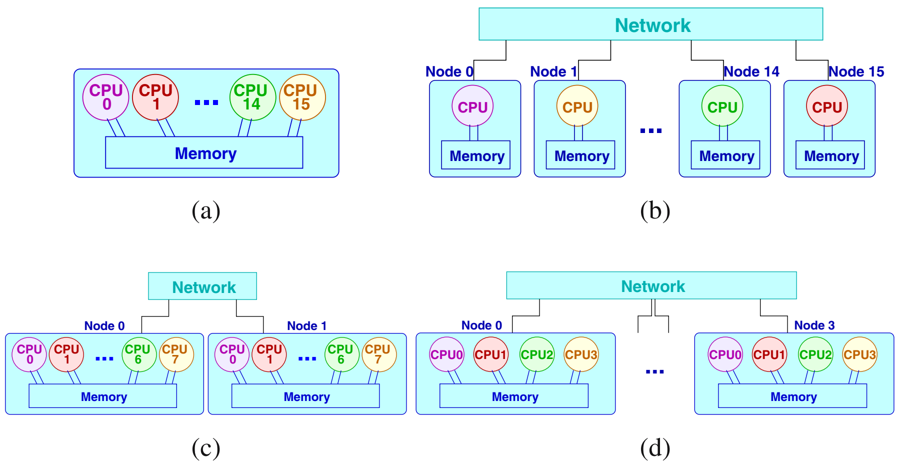


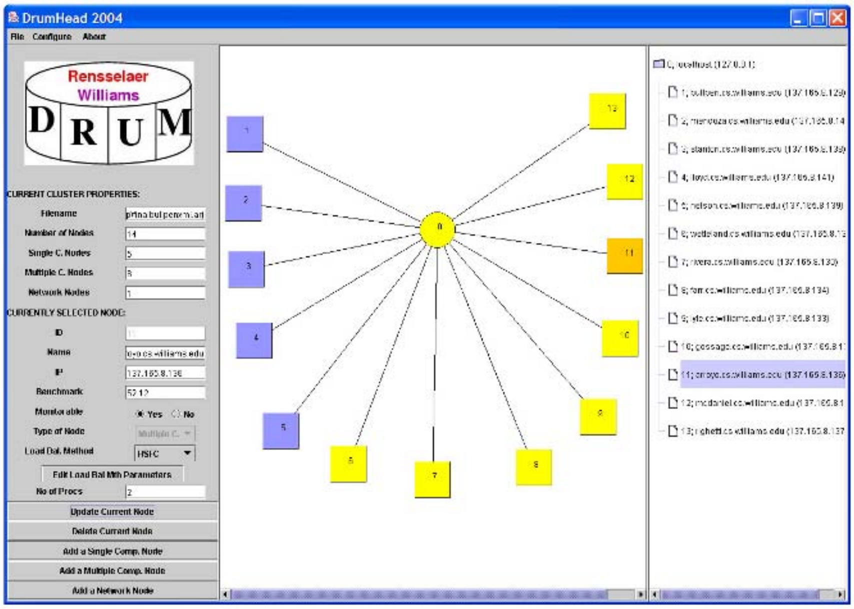
Fig. 2. Examples of parallel computing environments with 16 processors: (a) a 16-way SMP workstation; (b) a 16-node computer with all uniprocessor nodes, connected by a network; (c) two 8-way SMP workstations connected by a network; and (d) four 4-way SMP workstations connected by a network

$MLSI = 1.77$ and $GSI = 0.61$. This would be useful when the primary concern is a good computational balance, as in a shared-memory environment (Figure 2b). Using only ParMetis achieves excellent surface index values, $MLSI = 0.40$ and $GSI = 0.20$, but at the expense of a large computational imbalance, where partition sizes range from 49,389 to 89,302 regions. For a computation running on a network of workstations (NOW) (Figure 2b), it may be worth accepting the significant load imbalance to achieve the smaller communication volume.

For hierarchical systems, a hybrid partitioning may be desirable. Consider the two SMP configurations connected by a slow network as shown in Figure 2c,d. In the two 8-way node configuration (Figure 2c), ParMetis is used to divide the computation between the two SMP nodes, resulting in partitions of 532,063 and 570,955 regions, with $MLSI = 0.06$ and $GSI = 0.03$. Within each SMP, the mesh is partitioned eight ways using RIB, producing partitions within each SMP balanced to within one element, and with overall $MLSI = 1.88$ and $GSI = 0.56$. Since communication across the slow network is minimized, this is an appropriate partitioning for this environment. A similar strategy for the four 4-way SMP configuration (Figure 2d) results in a ParMetis partitioning across the four SMP nodes with 265,897, 272,976, 291,207 and 272,938 elements and $MLSI = 0.23$ and $GSI = 0.07$. Within each SMP, partitions are again balanced to within one element, with overall $MLSI = 1.32$ and $GSI = 0.32$. We first presented an example of this type of hybrid partition in [32], although the partitions presented therein were not generated by a fully automatic procedure.

Zoltan's hierarchical balancing automates the creation of such partitions. It can be used directly by an application or be guided by the tree representation of the computational environment created and maintained by the Dynamic Resource Utilization Model (DRUM) [10,13,34]. DRUM is a software system that supports automatic resource-aware partitioning and dynamic load balancing for heterogeneous, non-dedicated, and hierarchical computing environments. DRUM dynamically models the computing environment using a tree structure that encapsulates the capabilities and performance of communication and processing resources. The tree is populated with performance data obtained from *a priori* benchmarks and dynamic monitoring agents that run concurrently with the application. It is then used to guide partition-weighted and hierarchical partitioning and dynamic load balancing. Partition-weighted balancing is discussed further in [13]. DRUM's graphical configuration program (Figure 3) may be used to facilitate the specification of hierarchical balancing parameters at each network and multiprocessing node.

The hierarchical balancing implementation utilizes a lightweight "intermediate hierarchical balancing structure" (IHBS) and a set of callback functions. This permits an automated and efficient hierarchical balancing which can use any of the procedures available within Zoltan without modification and in any combination. Hierarchical balancing is invoked by an application in the same way as other Zoltan procedures. A hierarchical balancing step begins by building an IHBS using these callbacks. The IHBS is an augmented version of the distributed graph structure that Zoltan builds to make use of the ParMetis [21] and Jostle [36] libraries. The hierarchical balancing procedure then provides its own callback functions to allow existing Zoltan procedures to be used to query and update the IHBS at each level of a hierarchical balancing. After all levels



CURRENT CLUSTER PROPERTIES:
Filename: prina bulpen.xml
Number of Nodes: 14
Single C. Nodes: 5
Multiple C. Nodes: 8
Network Nodes: 1

CURRENTLY SELECTED NODE:
ID: 11
Name: arroyo.cs.williams.edu
IP: 137.165.8.136
Benchmark: 52.12
Monitorable: [✓ Yes] No
Type of Node: Multiple C.
Load Bal. Method: HSFC
No of Procs: 2

0, localhost (127.0.0.1)
1, bullpen.cs.williams.edu (137.165.8.129)
2, mendoza.cs.williams.edu (137.165.8.14
3, stanton.cs.williams.edu (137.165.8.138)
4, lloyd.cs.williams.edu (137.165.8.141)
5, nelson.cs.williams.edu (137.165.8.139)
6, wetteland.cs.williams.edu (137.165.8.13
7, rivera.cs.williams.edu (137.165.8.130)
8, tom.cs.williams.edu (137.165.8.134)
9, ybo.cs.williams.edu (137.165.8.133)
10, gossage.cs.williams.edu (137.165.8.1
11, arroyo.cs.williams.edu (137.165.8.136)
12, mcdaniel.cs.williams.edu (137.165.8.1
13, righetti.cs.williams.edu (137.165.8.137

Fig. 3. DRUM's graphical configuration program being used to edit a description of the "Bullpen Cluster" at Williams College. This program may be used to specify hierarchical balancing parameters for Zoltan

of the hierarchical balancing have been completed, Zoltan's usual migration arrays are constructed and returned to the application. Thus, only lightweight objects are migrated internally between levels, not the (larger and more costly) application data. Figure 4 shows the interaction between Zoltan and an application when hierarchical balancing is used.

3 Examples

We have tested our procedures using a software package called *LOCO* [16], which implements a parallel adaptive discontinuous Galerkin [3] solution of the compressible Euler equations. We consider the "perforated shock tube" problem, which models the three-dimensional unsteady compressible flow in a cylinder containing a cylindrical vent [14]. This problem was motivated by flow studies in perforated muzzle brakes for large calibre guns [12]. The initial mesh contains 69,572 tetrahedral elements. For these experiments, we stop the computation after 4 adaptive steps, when the mesh contains 254,510 elements.

Preliminary results are show that hierarchical partitioning can be competitive with the best results from a graph partitioner alone. We use two eight-processor computing environments: one with four Sun Enterprise 220R servers, each with two 450MHz Sparc UltraII processors, the other with two Sun Enterprise 420R servers, each with four

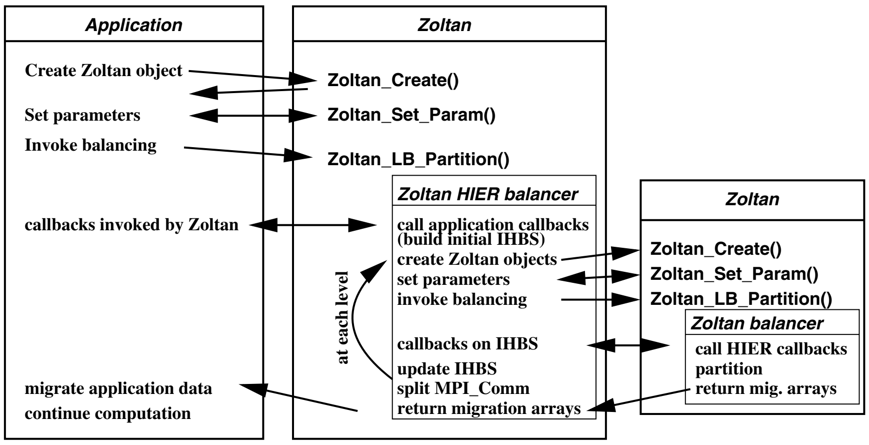


Fig. 4. Interaction between Zoltan and applications when hierarchical balancing is used

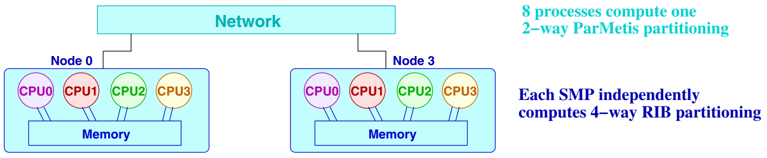


Fig. 5. Hierarchical balancing algorithm selection for two 4-way SMP nodes connected by a network

450MHz Sparc UltraII processors. The nodes are dedicated to computation and do not perform file service. In both cases, inter-node communication is across fast (100 Mbit) Ethernet. A comparison of running times for the perforated shock tube in these computing environments for all combinations of traditional and hierarchical procedures shows that while ParMetis multilevel graph partitioning alone often achieves the fastest computation times, there is some benefit to using hierarchical load balancing where ParMetis is used for inter-node partitioning and inertial recursive bisection is used within each node. For example, in the four-node environment (Figure 5), the computation time following the fourth adaptive step is 571.7 seconds for the hierarchical procedure with ParMetis and RIB, compared with 574.9 seconds for ParMetis alone, 702.7 seconds for Hilbert SFC partitioning alone, 1508.2 seconds for recursive coordinate bisection alone, and 822.9 seconds for RIB alone. It is higher for other hierarchical combinations of methods.

4 Discussion

We have demonstrated the ability to use the hierarchical balancing implemented within Zoltan as both an initial partitioning and a dynamic load balancing procedure for a realistic adaptive computation. While a traditional multilevel graph partitioning is often the most effective for this application and this computing environment, the results to date demonstrate the potential benefits of hierarchical procedures. In cases where the top-level multilevel graph partitioner achieves a decomposition without introducing load imbalance, it provides the fastest time-to-solution in our studies to this point (though only slightly faster than the multilevel graph partitioner alone is used). When imbalance is introduced by the multilevel graph partitioners, using hierarchical balancing to achieve strict balance within an SMP is beneficial.

Studies are underway that utilize hierarchical balancing on larger clusters, on other architectures, and with a wider variety of applications. We expect that hierarchical balancing will be most beneficial when the extreme hierarchies found in grid environments are considered. Significant benefits will depend on an MPI implementation that can provide a very efficient intra-node communication. This is not the case in the MPICH [17] implementation used on the cluster in our experiments. Here, all communication needs to go through a network layer (MPICH's p4 device), even if two processes are executing on the same SMP node. We are eager to run experiments of clusters built from other types of SMP nodes and on computational grids, and specifically those with MPI implementations that do provide appropriate intra-node communication optimizations.

Enhancements to the hierarchical balancing procedures will focus on usability and efficiency. Further enhancements to DRUM's machine model and graphical configuration tool will facilitate and automate the selection of hierarchical procedures. Efficiency may be improved by avoiding unnecessary updates to the intermediate structure, particularly at the lowest level partitioning step. Maintaining the intermediate structure across subsequent rebalancing steps would reduce startup costs, but is complex for adaptive problems. It would also be beneficial to avoid building redundant structures, such as when ParMetis is used at the highest level of a hierarchical balancing, however this would require some modification of individual Zoltan methods, which we have been careful to avoid thus far.

The IHBS itself has potential benefits outside of hierarchical balancing. It could be used to allow incremental enhancements and post-processing "smoothing" [15] on a decomposition before Zoltan returns its migration arrays to the application. The IHBS could also be used to compute multiple "candidate" decompositions with various algorithms and parameters, allowing Zoltan or the application to compute statistics about each and only accept and use the one deemed best.

Partitioning is only one factor that may be considered for an effective resource-aware computation. Ordering of computation and of communication, data replication to avoid communication across slow interfaces, and use of multithreading are other resource-aware enhancements that may be used. DRUM's machine model currently includes some information that may be useful for these other types of optimizations, and it will be augmented to include information to support others.

Acknowledgments

The authors were supported in part by Sandia contract PO15162 and the Computer Science Research Institute at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

The authors would like to thank Erik Boman, Karen Devine, and Bruce Hendrickson (Sandia National Laboratories, Albuquerque, USA) for their valuable input on the design of Zoltan's hierarchical balancing procedures.

Students Jin Chang (Rensselaer), Laura Effinger-Dean (Williams College), Luis Gervasio (Rensselaer), and Arjun Sharma (Williams College) have contributed to the development of DRUM.

We would like to thank the reviewers of this article for their helpful comments and suggestions.

References

1. S. B. Baden and S. J. Fink. A programming methodology for dual-tier multicomputers. *IEEE Transactions on Software Engineering*, 26(3):212–216, 2000.
2. M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Computers*, 36:570–580, 1987.
3. R. Biswas, K. D. Devine, and J. E. Flaherty. Parallel, adaptive finite element methods for conservation laws. *Appl. Numer. Math.*, 14:255–283, 1994.
4. E. Boman, K. Devine, R. Heaphy, B. Hendrickson, M. Heroux, and R. Preis. LDRD report: Parallel repartitioning for optimal solver performance. Technical Report SAND2004–0365, Sandia National Laboratories, Albuquerque, NM, February 2004.
5. C. L. Bottasso, J. E. Flaherty, C. Özturan, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. The quality of partitions produced by an iterative load balancer. In B. K. Szymanski and B. Sinharoy, editors, *Proc. Third Workshop on Languages, Compilers, and Runtime Systems*, pages 265–277, Troy, 1996.
6. T. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization". In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452. SIAM, 1993.
7. P. M. Campbell, K. D. Devine, J. E. Flaherty, L. G. Gervasio, and J. D. Teresco. Dynamic octree load balancing using space-filling curves. Technical Report CS-03-01, Williams College Department of Computer Science, 2003.
8. G. Cybenko. Dynamic load balancing for distributed memory multiprocessors. *J. Parallel Distrib. Comput.*, 7:279–301, 1989.
9. K. Devine, E. Boman, R. Heaphy, B. Hendrickson, and C. Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.
10. K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio. New challenges in dynamic load balancing. *Appl. Numer. Math.*, 52(2–3):133–152, 2005.
11. K. D. Devine, B. A. Hendrickson, E. Boman, M. St. John, and C. Vaughan. *Zoltan: A Dynamic Load Balancing Library for Parallel Applications; User's Guide*. Sandia National Laboratories, Albuquerque, NM, 1999. Tech. Report SAND99-1377. Open-source software distributed at `http://www.cs.sandia.gov/Zoltan`.
12. R. E. Dillon Jr. A parametric study of perforated muzzle brakes. ARDC Tech. Report ARLCB-TR-84015, Benét Weapons Laboratory, Watervliet, 1984.
13. J. Faik, J. D. Teresco, K. D. Devine, J. E. Flaherty, and L. G. Gervasio. A model for resource-aware load balancing on heterogeneous clusters. Technical Report CS-05-01, Williams College Department of Computer Science, 2005. Submitted to Transactions on Parallel and Distributed Systems.

14. J. E. Flaherty, R. M. Loy, M. S. Shephard, M. L. Simone, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Distributed octree data structures and local refinement method for the parallel solution of three-dimensional conservation laws. In M. Bern, J. Flaherty, and M. Luskin, editors, *Grid Generation and Adaptive Algorithms*, volume 113 of *The IMA Volumes in Mathematics and its Applications*, pages 113–134, Minneapolis, 1999. Institute for Mathematics and its Applications, Springer.
15. J. E. Flaherty, R. M. Loy, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz. Adaptive local refinement with octree load-balancing for the parallel solution of three-dimensional conservation laws. *J. Parallel Distrib. Comput.*, 47:139–152, 1997.
16. J. E. Flaherty, R. M. Loy, M. S. Shephard, and J. D. Teresco. Software for the parallel adaptive solution of conservation laws by discontinuous Galerkin methods. In B. Cockburn, G. Karniadakis, and S.-W. Shu, editors, *Discontinous Galerkin Methods Theory, Computation and Applications*, volume 11 of *Lecture Notes in Compuational Science and Engineering*, pages 113–124, Berlin, 2000. Springer.
17. W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996.
18. B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95*, 1995.
19. Y. F. Hu and R. J. Blake. An optimal dynamic load balancing algorithm. Preprint DL-P-95-011, Daresbury Laboratory, Warrington, WA4 4AD, UK, 1995.
20. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Scien. Comput.*, 20(1), 1999.
21. G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. *SIAM Review*, 41(2):278–300, 1999.
22. E. Leiss and H. Reddy. Distributed load balancing: design and performance analysis. *W. M. Kuck Research Computation Laboratory*, 5:205–270, 1989.
23. W. F. Mitchell. Refinement tree based partitioning for adaptive grids. In *Proc. Seventh SIAM Conf. on Parallel Processing for Scientific Computing*, pages 587–592. SIAM, 1995.
24. A. Patra and J. T. Oden. Problem decomposition for adaptive hp finite element methods. *Comp. Sys. Engng.*, 6(2):97–109, 1995.
25. J. R. Pilkington and S. B. Baden. Dynamic partitioning of non-uniform structured workloads with spacefilling curves. *IEEE Trans. on Parallel and Distributed Systems*, 7(3):288–300, 1996.
26. A. Pothen, H. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Mat. Anal. Appl.*, 11(3):430–452, 1990.
27. R. Rabenseifner and G. Wellein. Comparision of parallel programming models on clusters of SMP nodes. In H. Bock, E. Kostina, H. Phu, and R. Rannacher, editors, *Proc. Intl. Conf. on High Performance Scientific Computing*, pages 409–426, Hanoi, 2004. Springer.
28. M. S. Shephard, J. E. Flaherty, H. L. de Cougny, C. Özturan, C. L. Bottasso, and M. W. Beall. Parallel automated adaptive procedures for unstructured meshes. In *Parallel Comput. in CFD*, number R-807, pages 6.1–6.49. Agard, Neuilly-Sur-Seine, 1995.
29. H. D. Simon. Partitioning of unstructured problems for parallel processing. *Comp. Sys. Engng.*, 2:135–148, 1991.
30. C. A. Taylor, T. J. R. Hugues, and C. K. Zarins. Finite element modeling of blood flow in arteries. *Comput. Methods Appl. Mech. Engrg.*, 158(1–2):155–196, 1998.
31. V. E. Taylor and B. Nour-Omid. A study of the factorization fill-in for a parallel implementation of the finite element method. *Int. J. Numer. Meth. Engng.*, 37:3809–3823, 1994.
32. J. D. Teresco, M. W. Beall, J. E. Flaherty, and M. S. Shephard. A hierarchical partition model for adaptive finite element computation. *Comput. Methods Appl. Mech. Engrg.*, 184:269–285, 2000.

33. J. D. Teresco, K. D. Devine, and J. E. Flaherty. *Numerical Solution of Partial Differential Equations on Parallel Computers*, chapter Partitioning and Dynamic Load Balancing for the Numerical Solution of Partial Differential Equations. Springer-Verlag, 2005.
34. J. D. Teresco, J. Faik, and J. E. Flaherty. Resource-aware scientific computation on a heterogeneous cluster. *Computing in Science & Engineering*, 7(2):40–50, 2005.
35. J. D. Teresco and L. P. Ungar. A comparison of Zoltan dynamic load balancers for adaptive computation. Technical Report CS-03-02, Williams College Department of Computer Science, 2003. Presented at COMPLAS '03.
36. C. Walshaw and M. Cross. Parallel Optimisation Algorithms for Multilevel Mesh Partitioning. *Parallel Comput.*, 26(12):1635–1660, 2000.
37. M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proc. Supercomputing '93*, pages 12–21. IEEE Computer Society, 1993.
38. R. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency*, 3:457–481, October 1991.

Cache Optimizations for Iterative Numerical Codes Aware of Hardware Prefetching

Josef Weidendorfer and Carsten Trinitis

Technische Universität München, Germany
{weidendo,trinitic}@cs.tum.edu

Abstract. Cache optimizations use code transformations to increase the locality of memory accesses and use prefetching techniques to hide latency. For best performance, hardware prefetching units of processors should be complemented with software prefetch instructions. A cache simulation enhanced with a hardware prefetcher is presented to run code for a 3D multigrid solver. Thus, cache misses not predicted can be handled via insertion of prefetch instructions. Additionally, *Interleaved Block Prefetching* (IBPF), is presented. Measurements show its potential.

1 Introduction

Cache optimizations typically include code transformations to increase the locality of memory accesses: standard strategies are loop blocking and relayouting of data structures, e.g. splitting or joining of arrays and padding [12]. An orthogonal approach is to enable for latency hiding by introducing prefetching techniques; i.e., by ensuring that any data is loaded early enough before it is actually used. Software prefetching enables this by inserting cache load instructions into the program code. However, the use of such instructions consumes both decoding bandwidth and hardware resources for the handling of outstanding loads. Because of this and the added complexity of manually inserting prefetch instructions, modern processors like Intel Pentium 4, Pentium-M [10] or AMD Athlon, are equipped with hardware prefetch units which predict future memory accesses in order to load data into cache in advance.

Both prefetch approaches can be combined. Usually, there will be parts of code where hardware prefetching is doing a fine job, and parts where manual insertion of prefetch instructions is needed. The two cases can be distinguished by doing measurements with hardware performance counters. To this end, we use the Intel Pentium-M processor hardware, which has performance counters measuring the effect of its hardware prefetch unit [10]. This approach has some drawbacks: the actual prefetch algorithm implemented inside the processor is unknown, and thus, manual insertions of prefetch instructions will be specific to one processor. Therefore, additional measurements are done by simulating a known hardware prefetch algorithm. By choosing a simple stream detection, we are sure that program code where this prefetch algorithm is working fine, is also running fine with any existing hardware prefetcher. Additionally, we can compare results with and without hardware prefetching.

There is however a case where prefetching alone can not help: When performance of a code section is limited by available bandwidth to main memory, prefetching can not do any better. The first step to improve this situation is to use loop blocking [12]: instead of going multiple times over a large block of data not fitting into the cache, we split up this block into smaller ones fitting into the cache, and go multiple times over each small block before handling the next one. Thus, only in the first run over a small block has to fetch data from main memory. Still, this first run exhibits the bandwidth limitation. But as further runs on the small block do not use main memory bandwidth at all, this time can be used to prefetch the next block. Of course, the block size needs to be corrected in a way that two blocks fit into cache. We call this technique *Interleaved Block Prefetching*.

For this study, a 3D multigrid solver is used as application. The code is well known, and various standard cache optimizations were applied [11]. The main motivation for this work is the fact that standard cache optimizations do not work as well as expected on newer processors [17]. On the one hand, hardware prefetchers seem to help in the non-optimized case, on the other hand, they sometimes seem to work against manual cache optimizations: e.g. blocking with small loop intervals, especially in 3D, leads to a lot of unneeded activity from a hardware prefetcher with stream detection.

In the next chapter, some related work is presented. Afterwards, we give a survey of the cache simulator used and the hardware prefetch extension we added. We show measurements for our multigrid solver, using statistical sampling with hardware performance counters, and the simulator without and with hardware prefetching. Finally, for a simple example we introduce interleaved block prefetching and give some real measurements. Since adding interleaved block prefetching to the multigrid solver is currently work in progress, results will be given in a subsequent publication.

2 Related Work

Regarding hardware prefetchers, [3] gives a good overview of well known algorithms. Advanced algorithms are presented in [1]. Still, in hardware manuals of processors like Pentium-4 [10], there is only a vague description of the prefetch algorithm used.

Simulation is often used to get more details on the runtime behavior of applications or for research and development of processors [16],[9]. For memory access behavior and cache optimization, it is usually enough to simulate the cache hierarchy only like in MemSpy [14] or SIGMA [7]. The advantage of simulation is the possibility to get more useful metrics than plain hit/miss counts like reuse distances [4] or temporal/spatial locality [2].

For real measurements, we choose statistical sampling under Linux with OProfile [13]. With instrumentation, we would get exact values, e.g. using Adaptor [5], a Fortran source instrumentation tool, or DynaProf [8]. The latter uses DynInst [6] to instrument the running executable with low overhead. All these tools allow to use the hardware performance counters available on a processor.

3 Simulation of a Simple Hardware Prefetcher

Our cache simulator is derived from the cache simulator Cachegrind, part of the Valgrind runtime instrumentation framework [15], and thus, it can run unmodified executables.

We describe the simulator with its on-the-fly call-graph generation together with our visualization tool KCachegrind in more detail in [18]. With a conversion script, the latter is also able to visualize the sampling data from OProfile.

The implemented hardware prefetch algorithm gets active on L1 misses: for every 4KB memory page, it checks for sequential loads of cache lines, either upwards or downwards. When three consecutive lines are loaded, it assumes a streaming behavior and triggers prefetching of the cache line which is 5 lines in advance. As the simulator has no way to do timings, we simply assume this line to be loaded immediately. Although this design is arguable, it fulfills our requirement that every sensible hardware prefetcher should at least detect the same streaming as our prefetcher. The simulation will show code regions where simple hardware prefetchers probably have problems, i.e. where prefetching techniques are worthwhile.

The following results have been measured for a cache-optimized Fortran implementation of multigrid V(2,2) cycles, involving variable 7-point stencils on a regular 3D grid with 129^3 nodes. Compared to the standard version, the simulated as well as real measurement show little more than half the number of L2 misses / L2 lines read in. As only 2 smoothing iterations can be blocked with the given multigrid cycle, this shows that the blocking works quite well. In search for further improvement using prefetching, Table 1 shows the number of simulated L2 misses, first column with hardware prefetching switched off, second column with hardware prefetching switched on (i.e. only misses that are not catched by the prefetcher), and third column the number of prefetch actions issued. For real measurements, the Pentium-M can count L2 lines read in because of L2 misses alone, as shown in column 4, lines read in because of a request from the hardware prefetcher only in column 5. The last column shows the number of prefetch requests issued by the hardware. All numbers are given in millons of events (mega events). The first line gives a summary for the whole run, the next 2 lines splitted up by top functions: RB4W is the red-black gauss-seidel smoother, doing the main work, and RESTR is the restriction step in the multigrid cycle.

Simulation results show that the function RESTR is hardware-prefetcher friendly, as L2 misses go down by a large amount when the prefetcher is switched on in the simulation. In contrast, for the first function, RB4W, our simulated prefetcher does not seem to work. Looking at actual hardware, the prefetcher in the Pentium-M works better than the simulated one: it reduces the numbers of L2 misses (col. 4) even more than our one (col.2), especially it is doing a better job in RB4W. This shows that our simulation can show the same trend as will be seen in real hardware regarding friendliness to stream prefetching.

4 Interleaved Block Prefetching

Pure block prefetching means prefetching multiple sequential cache lines before actually using them afterwards. In contrast, interleaved block prefetching (IBPF) prefetches a block to be used in the future interleaved with computation on the previous block. One has to be careful to not introduce conflicts between the current and the next block.

Interleaved block prefetching is best used with the standard loop blocking cache optimization. When a program has to work multiple times on a huge array not fitting

Table 1. Measurements for the multigrid code [MEv]

	Simulated			Real		
	L2 Misses		Pref.	L2 Lines In		Pref.
	Pf. Off	Pf. On	Requests	due to Misses	due to Pref.	Requests
Summary	361	277	110	241	130	373
RB4W	233	226	-	201	47	270
RESTR	108	37	-	28	76	92

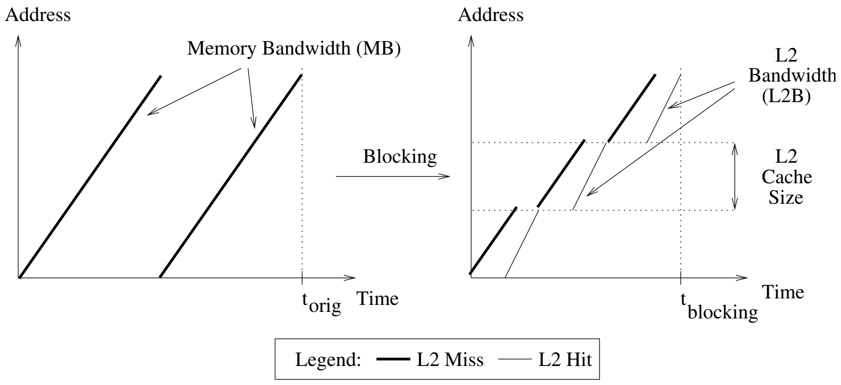


Fig. 1. The blocking transformation

into cache, data has to be loaded every time from main memory. When data dependencies of the algorithm allow to reorder work on array elements such that the multiple passes can be done consecutively on smaller parts of the array, blocking is possible. This improves temporal locality of memory accesses, i.e. accesses to the same data happen near to each other. By using block sizes that fit into cache (usually L2), this means that only for the first run over a small block, data has to be fetched from main memory. Fig. 1 depicts this scenario on a 1-dimensional array. The time $t_{blocking}$ is substantially less than t_{orig}: with blocking, the second iteration of each block can be done with L2 bandwidth $L2B$ instead of memory bandwidth MB, and thus gets a speedup of $\frac{MB}{L2B}$, i.e.

$$t_{blocking} = t_{orig} * (\frac{1}{2} + \frac{L2B}{2 * MB}).$$

As further runs on the small block do not use main memory bandwidth at all, this time can be used to prefetch the next block. This needs block size corrected in a way that two blocks fit into cache. For an inner block, we assume it to be already in the cache when work on it begins. When there are N runs over this block, we can interleave prefetching of the next block with computation over these N iterations. For the multigrid code, $N = 2$, i.e. the resulting pressure on memory bandwidth is almost cut in half. Fig. 2 shows this scenario. If we suppose that the needed bandwidth for prefetching is not near the memory bandwidth limit, almost all the time we can work with the L2 bandwidth

$L2B$, depending on the relation of the first block size to the whole array size as shown in the figure. Note that for the first block, memory pressure will be even higher than before because in addition to the first block, half of the second block is prefetched simultaneously.

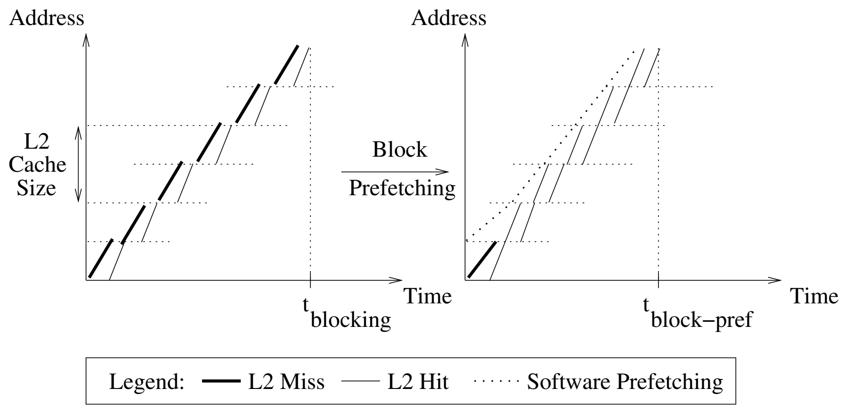


Fig. 2. The interleaved block prefetching transformation

In the following, a 1-dimensional streaming micro benchmark is used: it runs multiple times over an 8 MB-sized array of floating point values of double precision, doing one load and 1 or 2 FLOPS per element (running sums). Table 2 gives results with 1-dimensional blocking for different numbers of N. With pure blocking, $N = 1$ make no sense, as data is not reused. Still, block prefetching, which is the same as normal prefetching in this case, obviously helps. Again, these results were measured on a 1.4 GHz Pentium-M with DDR-266 memory, i.e. a maximum memory bandwidth of 2.1 MB/s. The benchmark was compiled to use x87 instructions only (no MMX or SSE).

For each benchmark and N, we show the runtime in seconds, and the achieved (netto) bandwidth from main memory. The netto bandwidth is calculated from the runtime and the known amount of data loaded. A brutto bandwidth that gives the bandwidth needed from memory to L2, can be calculated from memory read burst transactions, measured by a performance counter: the benchmark only loads data, and cache line loads are translated to burst requests. The brutto bandwidth is always between 50 and 100 MB/s higher than the netto bandwidth in this benchmark, and therefore not shown in the table.

Measurement shows that runtimes without IBPF can slow down by up to 46 percent compared to runtimes with IBPF ($N = 2$, 1 Flop/Load). To see the effect on the number of L2 lines read in on the one hand because of misses, and on the other hand because of hardware prefetch requests, we show these numbers for the first benchmark. Obviously, the prefetching virtually switches off the hardware prefetching.

One has to note that interleaving prefetching with regular computation can increase instruction overhead: in the micro benchmark, an additional loop nesting level had to be introduced for inserting one prefetch instruction each time $N * 8$ floating point elements are summed up: with a cache line size of 64 bytes, 8 double-precision values fit into

Table 2. Results of a micro benchmark

	Without Pref.				With IBPF			
N	1	2	3	10	1	2	3	10
1 Flop/Load								
Runtime [s]	5.52	**3**.92	3.45	2.73	4.89	**2**.67	2.50	2.44
MFlop/s	194	274	311	393	220	402	430	440
Netto [GB/s]	1.48	1.04	0.79	0.30	1.68	1.53	1.09	0.34
Lines In b/o Misses [MEv.]	4.2	2.1	1.4	0.4	134	67	45	13
Lines In b/o Pref. [MEv.]	130	65	44	13	0.45	0.54	0.27	0.15
2 Flops/Load								
Runtime [s]	7.20	5.24	4.81	3.23	6.86	4.29	3.60	2.84
MFlop/s	298	410	446	665	313	501	597	756
Netto [GB/s]	1.14	0.78	0.57	0.25	1.19	0.95	0.76	0.29

one cache line, and with blocking of N iterations, the prefetching stream advances with $1/N$ of the computation stream. In more complex cases or with other CPUs, the instruction increase can cancel the benefit of interleaved block prefetching. To reduce this effect, future ISAs (instruction set architectures) should introduce block prefetching instructions, minimizing the additional instruction overhead.

5 Conclusion and Future Work

In this paper, we have shown the usefulness of introducing a simple hardware prefetch algorithm into a cache simulator. By comparing a simulation with hardware prefetching switched off with the simulation of the same program run, but prefetching switched on, one can see the positions in the code where insertion of software prefetching instructions is useful. We extend this result by presenting the block prefetching technique which is able to lower the pressure on memory bandwidth.

Still, for simulation with hardware prefetching to be really worthful, visualization possibilities have to be enhanced: even source annotation can not differentiate between iterations of the same loop body, as only sums are given. But it appears useful to be able to distinguish at least the first few iterations of a loop, as it is expected that prefetch characteristics change here. Thus, more detailed context separation of profile data is needed. The interleaved block prefetching technique has to be integrated into real world applications like our multigrid solver.

References

1. M. Bekerman, S. Jourdan, R. Romen, G. Kirshenboim, L. Rappoport, A. Yoaz, and U. Weiser. Correlated Load-Address Predictors. In *Proceedings of the 26th International Symposium on Computer Architecture*, pages 54–63, May 1999.

2. E. Berg and E. Hagersten. SIP: Performance Tuning through Source Code Interdependence. In *Proceedings of the 8th International Euro-Par Conference (Euro-Par 2002)*, pages 177–186, Paderborn, Germany, August 2002.
3. Stefan G. Berg. Cache prefetching. Technical Report UW-CSE 02-02-04, University of Washington, Februar 2002.
4. K. Beyls and E.H. D'Hollander. Platform-Independent Cache Optimization by Pinpointing Low-Locality Reuse. In *Proceedings of International Conference on Computational Science*, volume 3, pages 463–470, June 2004.
5. T. Brandes. Adaptor. homepage. http://www.scai.fraunhofer.de/291.0.html.
6. B. Buck and J.K. Hollingsworth. An API for Runtime Code Patching. *The International Journal of High Performance Computing Applications*, 14:317–329, 2000.
7. L. DeRose, K. Ekanadham, J. K. Hollingsworth, and S. Sbaraglia. SIGMA: A Simulator Infrastructure to Guide Memory Analysis. In *Proceedings of SC 2002*, Baltimore, MD, November 2002.
8. Dynaprof Homepage. http://www.cs.utk.edu/ mucci/dynaprof.
9. H. C. Hsiao and C. T. King. MICA: A Memory and Interconnect Simulation Environment for Cache-based Architectures. In *Proceedings of the 33rd IEEE Annual Simulation Symposium (SS 2000)*, pages 317–325, April 2000.
10. Intel Corporation. IA-32 Intel Architecture: Software Developers Manual.
11. M. Kowarschik, U. Rüde, N. Thürey, and C. Weiß. Performance Optimization of 3D Multigrid on Hierarchical Memory Architectures. In *Proc. of the 6th Int. Conf. on Applied Parallel Computing (PARA 2002)*, volume 2367 of *Lecture Notes in Computer Science*, pages 307–316, Espoo, Finland, June 2002. Springer.
12. M. Kowarschik and C. Weiß. An Overview of Cache Optimization Techniques and Cache-Aware Numerical Algorithms. In U. Meyer, P. Sanders, and J. Sibeyn, editors, *Algorithms for Memory Hierarchies — Advanced Lectures*, volume 2625 of *Lecture Notes in Computer Science*, pages 213–232. Springer, March 2003.
13. J. Levon. OProfile, a system-wide profiler for Linux systems. Homepage: http://oprofile.sourceforge.net.
14. M. Martonosi, A. Gupta, and T. E. Anderson. Memspy: Analyzing memory system bottlenecks in programs. In *Measurement and Modeling of Computer Systems*, pages 1–12, 1992.
15. N. Nethercote and J. Seward. Valgrind: A Program Supervision Framework. In *Proceedings of the Third Workshop on Runtime Verification (RV'03)*, Boulder, Colorado, USA, July 2003. Available at http://developer.kde.org/˜sewardj.
16. V. S. Pai, P. Ranganathan, S. V. Adve, and T. Harton. An Evaluation of Memory Consistency Models for Shared-Memory Systems with ILP Processors. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 12–23, October 1996.
17. N. Thürey. Cache Optimizations for Multigrid in 3D. Lehrstuhl für Informatik 10 (Systemsimulation), Institut für Informatik, University of Erlangen-Nuremberg, Germany, June 2002. Studienarbeit.
18. J. Weidendorfer, M. Kowarschik, and C. Trinitis. A Tool Suite for Simulation Based Analysis of Memory Access Behavior. In *Proceedings of International Conference on Computational Science*, volume 3, pages 455–462, June 2004.

Computationally Expensive Methods in Statistics: An Introduction

Organizer: Wolfgang M. Hartmann

SAS Institute, Inc., Cary NC, USA

Assuming a twodimensional data set with N observations (rows) and n variables (columns) there are two types of large scale data which require intensive computational work:

1. $N >> n$: traditional statistical methods: many points in low or medium dimensional space; many observations but moderate number of variables; traditional ANOVA, $L2$, $L1$, and $L\infty$ regression; discriminance analysis etc. applications in traditional data mining.
2. $N << n$: newer statistical methods: few points in very high dimensional space; few observations N but many variables n; *curse of dimensionality*; applications in Chemometrics (spectra), microarray expression data (few chips but many genes); text mining and search engines (few terms in many documents).

In this minisymposium we were trying to concentrate on the second form of data, few points in highdimensional space, where new analysis methods are being developed.

Wolfgang Hartmann, SAS Institute Inc., Cary NC, opened the session with an introduction clarifying the difference between dimension reduction and variable selection methods in statistical analyses. A definition of the dimension reduction and variable selection problem in statistics is formulated which is based on work of McCabe (1984). A number of commonly known exploratory and predictive modeling methods in statistics and machine learning (data mining) are classified either as dimension reduction or variable selection methods.

Paul Somerville from the University of Central Florida in his presentation on *Stepdown FDR Procedures for Large Numbers of Hypotheses* introduced some new developments for a method of multiple comparisons in data with a large number of variables as with microarray data. In such large applications stepwise methods are used for computational efficiency but may suffer either from a number of false hypotheses or in a loss of statistical power. This work is still in progress and the talk highlighted some of the statistical problems with the authors stepwise approach compared to others already available. The new procedure seems to result in a reduced number of false rejections with a negligible reduction in power when the expected number of false hypotheses is small.

Ulrich Mansmann, then from the University of Heidelberg, Department of Medical Biometry and Informatics, now at the University of Munich, proceeded with his talk on *Reproducible Statistical Analysis in Microarray Profiling Studies*. It is well known that for more complicate statistical modeling published results are not easily reproducable. Due to the complexity of the algorithms, the size of the data sets, and the limitations of the medium printed paper it is usually not possible to report all the minutiae of the data processing and statistical computations. Microarrays are a recent biotechnology that offer the hope of improved cancer classification, providing clinicians with the information to choose the most appropriate form of treatment. But, reproducing published results in this domain is harder than it may seem. For example, Tibshirani and Efron (2002) report: "We reanalysed the breast cancer data from van't Veer et al. (2002). ... Even with some help of the authors, we were unable to exactly reproduce this analysis.". To achieve reproducible calculations and to offer an extensible computational framework the tool of a *compendium* (Sawitzki , 1999; Leisch, 2002; Gentleman and Temple Lang, 2004; Sawitzki, 2002) is discussed. A *compendium* is a document that bundles primary data, processing methods (computational code), derived data, and statistical output with the textual documentation and conclusions. Ulrich's talk presents examples, discusses the problems hidden in the published analyses and demonstrates a strategy to improve the situation which is based on the vignette technology available from the R and Bioconductor projects (Ihaka and Gentleman, 1996; Gentleman and Carey, 2002).

Frank Bretz, then at the University of Hannover, now at *Novartis Pharmaceuticals* in Basel, Switzerland, presented some work with P. Westfall, Texas Tech University, on the computationally difficult problem of applying *multiple comparisons* to the analysis of microarray data. Algorithms for multiple comparisons usually require the highdimensional quadrature of multivariate normal or t distribution.

Analytic formulas were developed for various types of power and error rates of some closed testing procedures. The formulas involve non-convex regions that may be integrated with high, pre-specified accuracy using available software. The non-convex regions are represented as a union of hyper-rectangles. These regions are transformed to the unit hypercube, then summed, to create an expression for power that is the integral of a function defined on the unit hypercube. This function is then evaluated using existing quasi-Monte Carlo methods that are known for their accuracy. Applications include individual, average, complete, and minimal power, for closed pairwise comparisons (max T-based), as well as fixed sequence and non-pairwise comparisons.

Thomas Ragg, co-founder of *quantiom bioinformatics GmbH*, `http://www.quantiom.de`, of a small but growing company which "develops customized Analysis Strategies and specialized software solutions for bioanalytical questions and offers the necessary consulting, support, training, and service" concluded the session with a talk on *Normalization, variable ranking and model selection for high-dimensional genomic data - Design and Implementation of automated analysis strategies* which outlined the procedure for analyzing microarray data as it is commonly encountered by his team. Some crucial problems in building models based on empirical genomic data are

- normalization within experiments and between experiments to make them comparable
- filtering of irrelevant variables to reduce the complexity of the search space
- selection of appropriate subset of features (e.g. genes)
- model training based on the available data, where the model complexity needs to be balanced against the training error
- model selection based on a fitness criterion (e.g. model evidence)

- evaluation of the strategy and appropriate presentation of results

The analysis strategy was demonstrated for oligo microarray data and showed the benefits of the *quantiom* software implementation (in Java) with its modular design principle.

For references see the contributions.

Dimension Reduction vs. Variable Selection

Wolfgang M. Hartmann

SAS Institute, Inc., Cary NC, USA

Abstract. The paper clarifies the difference between dimension reduction and variable selection methods in statistics and data mining. Traditional and recent modeling methods are listed and a typical approach to variable selection is mentioned. In addition, the need for and types of cross validation in modeling is sketched.

1 Introduction

The purpose of the paper is an approach to highlight the difference between methods of variable selection (VS) and dimension reduction (DR) in statistics and to show some basic features of the most common methods. Both types of methods originally submerged in computational statistics and are now more and more used and developed in the area of *data mining*, sometimes called as *machine learning*. It is first necessary to explain some of the terms used in later parts of this paper.

The machine learning literature differs between two essential types of analysis:

Unsupervised Learning also called exploratory analysis methods: The analysis data set consists only of one set of variables $\mathbf{X}$ for which N measurements (observations, cases) are available. A modeling method is used to find similarities and significant differences either
- among the variables in $\mathbf{X}$
- or among the observations in $\mathbf{X}$.

Supervised Learning also called as predictive modeling: The analysis data set consists of two sets of variables, a set of response (dependent) variables $\mathbf{Y}$ and a set of predictor (independent) variables $\mathbf{X}$. A method of linear or nonlinear parametric modeling or a method of nonparametric modeling is used that establishes a relationship among the values of the $\mathbf{X}$ variables and allows to predicts the values of the $\mathbf{Y}$ variables with sufficient precision.

Here we will mostly deal with parametric modeling.

2 Difference Between DR and VS

The following notation outlines the difference between DR and VS and is based on work of McCabe (1984, [21]):

1. Given an $N \times n$ matrix $\mathbf{X}$ as training data.
2. Usually $\mathbf{X}$ is column centered around mean (sometimes a covariance matrix) or even scaled (sometimes a correlation matrix).

3. Find $n \times k$, $k \ll n$, transformation matrix $\mathbf{A}_k$
$$\mathbf{V} = \mathbf{X}\mathbf{A}_k, \text{ usually s.t. } \mathbf{A}_k^T \mathbf{A}_k = \mathbf{I}_k$$
4. For dimension reduction $\mathbf{A}_k$ is dense.
5. For variable selection: $\mathbf{A}_k = perm([\mathbf{I}_k : \mathbf{0}_{k\times(n-k)}]^T)$ where $perm(\mathbf{A})$ is row permuted $n \times k$ matrix.

This definition does not refer to the criterion for the selection of $\mathbf{A}_k$ and therefore covers both, supervised and unsupervised methods, of a wide range. It also does not specify the criterion used for selecting k, the number of dimensions or selected variables. Therefore, methods for DR and VS are of a very wide range.

From that follows that we understand variable selection as a discrete process (sparse notation) whereas dimension reduction is a continuous process (dense notation). Therefore, DR shows the grouping of variables, variables with low impact have smaller weights than those with large impact. On the other hand VR picks usually only one representative from each cluster of similar variables and does not refer to variables which may be very similar to those picked but with only slightly smaller impact. To find the highly correlated variables of those variables picked with VS usually postprocessing of the data is needed. As an implication of the discrete selection process of VS we can expect rather high variance in bootstrap and cross validation, compared to a much lower variance for the more continuous DR. That again has an impact on choosing the methods used for tuning hyper parameters of the model.

3 Overview on Methods for VS and DR

In the following we try to list the most common methods for VS and DR and refer to software found in the free statistical software *R* (see [26]) the authors own interactive matrix language CMAT (Hartmann, [15]) and in SAS/STAT (statistics)([29]), SAS/IML (interactive matrix language), and SAS/EM (enterprise miner). Of course, many other software packages like SPSS and S-PLUS are available that cover most the listed methods.

3.1 Methods for Dimension Reduction

The purpose is finding $p \ll n$ components/factors/clusters which are (linear or nonlinear) functions of the n variables which describe the *essential* features of the data in some sense (maximizing explained variance, de-noise data in some sense, sufficient for prediction).

1. Exploratory Modeling (Unsupervised Learning):
   (a) (Kernel) Dense Principal Components and Factor Analysis, e.g. see Harman (1976, [14]) and Mulaik (1972, [23])
      – functions `prcomp` and `princomp` and the `kernlab` package in R
      – functions `eig`, `nlkpca`, `factor`, and `sem` in CMAT
      – functions `eigval`, `eigvec`, and `eigen` in SAS/IML and PROCs PRINCOMP, FACTOR, and CALIS in SAS/STAT

   (b) Singular Value Decomposition and Correspondence Analysis, e.g. see Gifi (1981, [11]) and Greenacre (1984, [12])
      - function `svd` and functions `corresp`, `mca`, and `CoCoAn` in R
      - functions `svd`, `gsvd`, `arpack`, and `svdtrip` in CMAT; correspondence analysis currently not in CMAT
      - function `svd` in SAS/IML and PROC CORRESP in SAS/STAT
   (c) Multidimensional Scaling, e.g. see Carroll & Arabie (1998, [6]),
      - function `smdscale` in R
      - currently not in CMAT
      - PROC MDS in SAS/STAT
   (d) Methods of fuzzy variable clustering, e.g. see Kaufman & Rousseeuw (1990, [17])
      - function `cluster`, `hclust`, `mclust`, `cclust` in R
      - functions `cluster` and `emclus` in CMAT
      - PROCs ACECLUS, CLUSTER, and MODECLUS in SAS/STAT
2. Predictive Modeling (Supervised Learning):
   (a) (Kernel) Partial least-squares, e.g. see Wold (1966, [38]), Rosipal & Trejo (2001, [28])
      - package `plspcr` and `gpls` in R
      - functions `pls` and `nlkpls` in CMAT
      - PROC PLS in SAS/STAT (kernel PLS currently not in SAS)
   (b) Sliced inverse regression (SIR) and principal Hessian directions (pHd), e.g. see Weisberg, 2002, [35])
      - function `sir` in R
      - function `sir` in CMAT
      - currently not in SAS
   (c) Neural Networks (Ripley, 1994, [27]) and Support Vector Machines (Vapnik, 1995, [37]; Schoelkopf & Smola, 2000, [30])
      - function `nnet`, and function `svm` in `e1071` package and the `kernlab` package in R
      - function `svm` in CMAT
      - SAS PROCs SVM, NEURAL, and DMNEURL in SAS/EM

3.2 Methods for Variable Selection

The purpose is finding a subset of $p \ll n$ variables from the original data which describe the *essential* features of the data in some sense (maximizing explained variance, de-noise data in some sense, sufficient for prediction).

1. Exploratory Modeling (Unsupervised Learning):
   (a) Coloring a correlation matrix
   (b) Sparse Principal Components (SPCA) (Zou & Hastie, 2004, [41])
      - not in R
      - function `spca` in CMAT
      - not in SAS
   (c) Principal Variables (McCabe, 1984, [21])
   (d) Methods of discrete variable clustering

   i. using common cluster methods with the transposed data matrix, e.g. see Kaufman & Rousseeuw (1990, [17])
   ii. using hybrids of factor analysis and rotation to sparse factor patterns e.g. see Harman (1976, [14]) or Anderberg (1973, [1])
      – not in R
      – function `varclus` in CMAT
      – SAS PROC VARCLUS
   iii. K Median Clustering (Mangasarian & Wild, 2004, [20])

2. Predictive Modeling (Supervised Learning):
   (a) Type III analyses of many statistical modeling methods
   (b) Subset selection in (generalized linear) regression (Miller, 2002, [22]) : specifically stepwise (linear, logistic) regression
      – function `step`, `glm`, `leaps` in R
      – functions `lrforw`, `reg`, `glim`, `glmixd` in CMAT
      – PROCs REG, MIXED, LOGISTIC, PHREG, and GENMOD in SAS/STAT and PROC DMINE in SAS/EM
   (c) Recursive Partitioning and Regression Trees (CART, CHAID) see Breiman et.al (1984, [4])
      – package `tree` and functions `rpart`, `knntree` in R
      – function `split` in CMAT
      – PROCs SPLIT, DMSPLIT, and ARBOR in SAS/EM
   (d) Step-up and step-down Multivariate Testing, e.g. see Benjamini & Hochberg (1995, [2]) Somerville & Bretz (2001, [32]) Somerville (2004, [31])
      – function `multcomp` and `multtest` in Bioconductor package in R
      – function `multtest` in CMAT
      – PROC MULTTEST ins SAS/STAT
   (e) *Garotte* by Breiman (1993, [5])
      – currently not in R
      – function `garotte` in CMAT
      – currently not in SAS
   (f) (Univariate) Soft Thresholding (UST) by Chen, Donoho, & Saunders (1999, [7]), Tibshirani, Hastie, Narasimhan, & Chu (2002, [34])
   (g) *Lasso* by Tibshirani (1996, [33]) and Osborne, Presnell, & Turlach (2000, [24]) and *LARS* by Efron, Hastie, Johnstone, & Tibshirani (2002, [9])
      – functions `lars` and `lasso` in R
      – function `lars` in CMAT
      – currently not in SAS
   (h) *Elastic Net* by Zou and Hastie (2003, [40])
      – currently not in R
      – function `lars` in CMAT
      – currently not in SAS
   (i) Sparse $L1$ SV Regession (Bi, Bennett, Embrechts, Breneman, & Song, 2002, [3])

   - currently not in R
   - function `svm` in CMAT
   - currently not in SAS
- (j) Sparse $L1$ SV Classification (Fung & Mangasarian, 2003, [10])
   - currently not in R
   - function `svm` in CMAT
   - PROC SVM in SAS/EM
- (k) SVM Feature Selection by Guyon et.al. (2002, [13]), Weston et.al. (2000, [36])
   - currently not in R
   - function `svm` in CMAT
   - PROC SVM in SAS/EM
- (l) Feature Selection using Genetic Algorithms, e.g. see Yang & Honavar (1997, [39])
   - functions `gap`, `rgenoud`, and `gafit` in R
   - function `nlp` in CMAT
   - currently not in SAS
- (m) Bayesian methods of variable selection
   - `naiveBayes` in `e1071` package and the `BsMD` package in R
   - currently not in CMAT
   - currently not in SAS

Note, even though some of the software has the same name in CMAT and SAS, the implementations are usually very different. For example, the algorithms for variable clustering in CMAT are not necessarily based on a covariance or correlation matrix (which is for $n \ll N$ sometimes difficult to store) as with the PROC VARCLUS in SAS/STAT.

3.3 Example: Variable Selection with SV Regression

Illustrating the five-step *variable selection* algorithm by Bi, Bennett, Embrecht, Breneman, & Song (2001, [3]):

1. Attach a small set of random generated *gauge* variables (columns) to data set which have approximately zero correlation with response y.
2. Outer iteration performs *Bootstrap aggregation* (*Bagging*): using bootstrap samples for training data and average about results (reducing the variance, makes results more stable especially w.r.t. local optima of internal pattern search)
3. Pattern search cycle for hyper parameters of linear SV regression, regularization C and tube parameter ϵ is performed and best solution is selected for nearly optimal values of (C, ϵ).
4. The pattern search is based on K fold cross validation result. This CV cycle requires to optimize K LPs (*linear* support vector machine problems) for specific values of (C, ϵ).
5. Based on the former four steps, a small set of variables is selected. Using the small set of variables the last two steps of pattern search with cross validation is now performed for nonlinear kernel. That means, in addition to regularization C and tube parameter ϵ the pattern search now includes additional kernel parameters.

4 Cross Validation (CV)

The machine learning literature refers to three types of data sets that are useful for modeling:

Training Data: This data set contains N observations with n variables. For predictive modeling the latter can be predictor or response variables. This data set is being used to build the model.

Validation Data: This data set contains N_{val} observations at the same n variables as the training set is using. It may be used as a second data set at the modeling process to measure the goodness-of-fit of the model and to determine the termination of the modeling process. Using a validation data set is similar to cross validation and can increase the generalizability of the model and to reduce the possibility of *overfiting*. That means the resulting model will also perform well with other than the training data.

Test Data: This data set contains N_{test} observations and is usually not available at the modeling process. Even for predictive modeling it must not contain the response variables, but must contain all the predictive variables as the training data set. After a parameteric model was established using training and perhaps validation data, the model could be applied to the test data. In predictive modeling that means using the model to compute the expected values of response variables (scoring).

The purpose of cross validation is to:

1. **Increase generalizeability** of model estimates: model does not only fit one single sample, it also fits other samples of the same population well.
2. **Prevent overfitting**: too many parameters, some describing data noise specific for the sample used in training.

This is achieved by evaluating the model on data which do not belong to data used for training the model.

Block CV: in K-fold *block* CV there are K training runs: block of N/K observations is left out from model training; but the remaining $N - N/K$ observations are scored; e.g. left out for K=10 and N=100: i=1,...,10; i=11,...,20; i=21,...,30;... (biased for sorted data)

Split CV: in K-fold *split* CV there are K training runs: "successive groups of widely separated observations are held out as the test set", e.g. left out for K=10: i=1,11, 21,...; i=2,12,22,...; i=3,13,23,...

Random CV: in K-fold *random* CV there are K training runs: in each run N/K randomly selected observations are left out from model training; observations are selected from remaining pool, i.e. after K training runs each observation is left out exactly once;

Loo CV: in each of N training runs one observation is left out from the training data set: the remaining $N - 1$ observations are used for parm estimation, but the left out observation is scored (here is $K = N$). This is related to the well known *Jackknife* in statistics.

Test Set Validation: the model is trained (parameters estimated) based on training data; model is evaluated on test data set; assumes that there are two large enough data sets.

Random Validation: inside a loop of multiple training runs: the input data set is divided randomly in training and validation data set. The model estimates are computed with the training set and the fit is evaluated on the validation data set. The best model is selected based on the fit with the validation data sets. (If not done "balanced", the result can be biased, i.e. some observations are scored more than others)

A number of functions in CMAT, like `pls` and `svm` offer options for cross validation. Usually, Loo is computationally the most expensive method since it needs the large number of N estimations. But using an estimation method that exploits good starting values the difference in computer time compared to 10-fold cross validation can be surprisingly small. An advantage of Loo is that it can be used for sensitivity analysis and the detection of model outliers:

1. During the N leave-one-out runs, the model fit can be evaluated at the remaining $N - 1$ observations of the training set.
2. If the training model fit *improves* significantly when leaving out observation X_i, it indicates that observation X_i is a "model misfit", a model outlier.

References

1. Anderberg, M.R. (1973), *Cluster Analysis for Applications*, New York: Academic Press, Inc.
2. Benjamini, Y. & Hochberg, Y. (1995), “Controlling the false discovery rate: a practical and powerful approach to multiple testing”, *J. R. Statist. Soc.*, Ser. B, 289-300.
3. Bi, J., Bennett, K., Embrechts, M., Breneman, C. & Song, M. (2002), “Dimensionality Reduction via Sparse Support Vector Machines”, *Journal of Machine Learning Research* **1** 1-48.
4. Breiman, L., Friedman, J.H., Olshen, R.A. & Stone, C.J. (1984), *Classification and Regression Trees*, Wadsworth.
5. Breiman, L. (1993), “Better subset selection using the non-negative garotte”; *Technical Report*, Univ. of California, Berkeley.
6. Carroll, J.D. & Arabie, P. (1998), *Multidimensional scaling*, in M.H. Birnbaum (Ed.), *Handbook of Perception and Cognition: Measurement, Judgment and Decision Making*, p. 179-250, San Diego, CA: Academic Press.
7. Chen, S.S., Donoho, D.L., & Saunders, M. (1999), “Atomic decomposition by basis pursuit”, *SIAM Journal on Scientific Computing*, **20**, 33-61.
8. Donoho, D., Johnstone, I., & Tibshirani, R. (1995), “Wavelet shrinkage: asymptotia? (with discussion)”; *J. R. Statist. Soc.*, Ser. B, **57**, 301-337.
9. Efron, B., Hastie, T., Johnstone, I. & Tibshirani, R. (2002), “Least Angle Regression”, *The Annals of Statistics*, **32**, 407-499.
10. Fung, G. & Mangasarian, O.L. (2003), “A Feature Selection Newton Method for Support Vector Machine Classification”, *Computational Optimization and Aplications*, 1-18.
11. Gifi, A. (1981), *Nonlinear Multivariate Analysis*, Dep. of Data Theory, Univ. of Leiden.
12. Greenacre, M.J. (1988), *Theory and Applications of Correspondence Analysis*, London: Academic Press.
13. Guyon, I., Weston, J., Barnhill, S. & Vapnik, V. (2002), "Gene Selection for cancer classification using support vector machines"; *Machine Learning*, **46**, 389-422.

14. Harman, H.H. (1976), *Modern Factor Analysis*, Chicago: University of Chicago Press.
15. Hartmann, W. (1995), *CMAT Users Manual*, see `http://www.cmat.pair.com/cmat`
16. Joachims, T. (1999), "Making large-scale SVM learning practical", in B. Schölkopf, C.J.C. Burges, and A.J. Smola (eds), *Advances in Kernel Methods: Support Vector Learning*, Cambridge: MIT Press.
17. Kaufman, L. & Rousseeuw, P. (1990), *Finding Groups in Data: An Introduction to Cluster Analysis*, New York: Wiley.
18. Li, K.C. (1991), "Sliced inverse regression for dimension reduction"; *JASA*, **86**, 316-342.
19. Li, K.C. (1992), "On principal Hessian directions for data visualization and dimension reduction"; *JASA*, **87**, 1025-1034.
20. Mangasarian, O.L. & Wild, E.W. (2004), "Feature Selection in k-Median Clustering", Technical Report 04-01, Data Mining Institute, Madison: University of Wisconsin.
21. McCabe, G.P. (1984), "Principal variables"; *Technometrics*, **26**, 139-144.
22. Miller, A. (2002), *Subset Selection in Regression*, CRC Press, Chapman & Hall.
23. Mulaik, S.A. (1972), *The Foundations of Factor Analysis*, New York: Mc Graw Hill.
24. Osborne, M.R., Presnell, B., & Turlach, B.A. (2000), "On the LASSO and its Dual", JCGS, **9**, 319-337.
25. Osborne, M.R., Presnell, B., & Turlach, B.A. (2000), "A new approach to variable selection in least squares problems", *IMA Journal of Numerical Analysis*, **20**, 389-404.
26. *R* Language and packages see: `http://www.r-project.org/` and `http://cran.r-project.org/`
27. Ripley, B,D. (1996), *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press.
28. Rosipal, R. & Trejo, L.J. (2001), "Kernel partial least squares regression in reproducing kernel Hilbert space", *Journal of Machine Learning Research*, **2**, 97-123.
29. *SAS/STAT User's Guide*, (1990), Version 6, Second Printing, SAS Institute Inc., Cary, NC.
30. Schölkopf, B. & Smola, A.J. (2002), *Learning with Kernels*, Cambridge MA and London: MIT Press.
31. Somerville, P.N. (2004), "Step-down FDR Procedures for large numbers of hypotheses", this volume.
32. Somerville, P.N. & Bretz, F. (2001), "FORTRAN 90 and SAS-IML programs for computation of critical values for multiple testing and simultaneous confidence intervals", *Journal of Statistical Software*.
33. Tibshirani, R. (1996), "Regression shrinkage and selection via the Lasso", *J. R. Statist. Soc., Ser. B*, **58**, 267-288.
34. Tibshirani, R., Hastie, T., Narasimhan, B. & Chu, G. (2002), "Diagnosis of multiple cancer types by shrunken centroids of gene expression"; *Proceeding of the National Academy of Sciences*, **99**, 6567-6572.
35. Weisberg, S. (2002), "Dimension reduction regression with R", *JSS*, **7**.
36. Weston, J., Mukherje, S., Chapelle, O., Pontil, M., Poggio, T., & Vapnik, V. (2000), "Feature Selection for SVMs", *Neural Information Processing Systems*, **13**, 668-674.
37. Vapnik, V.N. (1995), *The Nature of Statistical Learning*, New York: Springer.
38. Wold, H. (1966), "Estimation of principal components and related models by iterative least squares", *Multivariate Analysis*, New York: Academic Press.
39. Yang, J. & Honavar, V. (1997), "Feature selection using a genetic algorithm", Technical Report, Iowa State University.
40. Zou, H., & Hastie, T. (2003), "Regression shrinkage and selection via the elastic net, with applications to micro arrays", Technical Report, Stanford University.
41. Zou, H., Hastie, T. & Tibshirani, R. (2004), "Sparse principal component analysis", Technical Report, Stanford University.

Reproducible Statistical Analysis in Microarray Profiling Studies

Ulrich Mansmann[1], Markus Ruschhaupt[2], and Wolfgang Huber[2]

[1] University of Heidelberg, Department for Medical Biometry and Informatics
INF 305, 69120 Heidelberg, Germany
mansmann@imbi.uni-heidelberg.de

[2] German Cancer Research Center, Division of Molecular Genome Analysis
INF 580, 69120 Heidelberg, Germany

Abstract. Reproducibility of calculations is a longstanding issue within the statistical community. Due to the complexity of the algorithms, the size of the data sets, and the limitations of the medium printed paper it is usually not possible to report all the minutiae of the data processing and statistical computations. Like the critical assessment of a mathematical proof it should be possible to check the software behind a complex data analysis. To achieve reproducible calculations and to offer an extensible computational framework the tool of a compendium is discussed.

1 Introduction

Microarray technology allows simultaneous measurement of thousands of transcripts within a homogeneous sample of cells [1]. It is of interest to relate these expression profiles to clinical phenotypes to improve the diagnosis of diseases and prognosis for individual patients [2]. A number of publications presented clinically promising results by combining this new kind of biological data with specifically designed algorithmic approaches. A selection out of these papers will be discussed [3,4,5,6] with respect to different aspects of reproducibility.

The most evident aspect of reproducibility is that of reproducing a calculation on the same data. Reproducing published results in the domain of microarray profiling studies is harder than it may seem. As example we look at a study of van 't Veer et al. [3] which was reanalysed by Tibshirani and Efron [7]. Both state in their paper: *We re-analyzed the breast cancer data from van 't Veer et al. ... Even with some help of the authors, we were unable to exactly reproduce this analysis.*

We do not know any example where classification results gained with one microarray technology and a special algorithm were reproduced using an alternative microarray platform and algorithm. Papers with diverging results on profiles for the prognosis of tumour recurrence for breast cancer patients are [3,4]. How does this observation relate to the idea of a common underlying disease process? Should profiles have something common which are developed for the same disease? Is it of significance if they do not?

The confounding of algorithmic problems with biotechnology and biology creates a gordic knot. The paper applies the tool of a *compendium* as interactive strategy to settle the algorithmic backbone of a profiling study and to derive reproducible results with a state-of-the-art methodology. A compendium is a document that bundles primary data, processing methods (computational code), derived data, and statistical output with the textual documentation and conclusions. It is interactive in the sense that it allows to modify the processing options, plug in new data, or insert further algorithms and visualisations.

2 Examples and Questions

2.1 Example 1

Van 't Veer et al. [3] classify breast cancer patients after curative resection with respect to the risk of tumour recurrence. The study includes 78 patients. Forty four patients had a *good prognosis* and did not suffer from a recurrence during the first 5 years after resection. Thirty four patients had a bad prognosis and experienced a recurrence during the first 5 years after resection. Agilent microarray technology was used to quantify the transcripts probed by 24,881 oligonucleotides. Additional prognostic factors like tumour grade, ER status, PR status, tumour size, patient age, angioinvasion were also documented. The authors were interested to develop a classifier based on the gene expression and to compare the relevance of the genomic signature to the prognostic value of standard clinical predictors. They used to following algorithm to establish the signature which contains 70 genes:

1. Starting with 24,881 genes, filtering on fold-change and a p-value criterion reduced the number of relevant genes to 4,936.
2. Based on an absolute correlation of at least 0.3 between gene expression and group indicator (0,1) a further reduction on 231 genes
3. Calculation of the 231 dimensional centroid vector for the 44 good prognosis cases.
4. Correlation of each case with this centroid is calculated, cut-off of 0.38 is chosen to exactly misclassify 3 with poor prognosis
5. Case is classified to good prognosis if correlation calculated for some number n of genes ($1 \leq n \leq 231$) with the centroid vector is ≥ 0.38, otherwise the case is classified to the *bad prognosis* group.
6. Starting with the top 5, 10, 15,... genes, classification procedure is carried out with leave-one-out cross-validation, to pick the optimal number of genes $\rightarrow$ 70

Based on this algorithm van 't Veer et al. achieved a correct classification for 26 of 44 patients without recurrence and for 31 of 34 with recurrence. Tibshirani and Efron [7] tried to reproduce this results and report: *Even with some help of the authors, we were unable to exactly reproduce this analysis*. In fact, the differences between both calculations were not dramatic. But, the example shows that algorithmic reproducibility is not a trivial issue and may have subjective elements. The algorithm is quite popular and is also used in [5] and other papers.

The van 't Veer examples rises the following questions: Why was a heuristic classification algorithm chosen and not a standard algorithm from machine learning? Are its computational aspects well understood? How important is the choise of the parameters for correct classification? Is the result easy to interpret? What justifies the popularity of the algorithm? Is the leave-one-out CV strategy appropriate? What is the effect of other CV strategies on the classification result?

2.2 Example 2

Huang et al. [4] investigated primary tumour samples from 52 patients with breast tumours and 1-3 positive lymph nodes. 18 patients had a recurrence within three years after surgery, and 34 patients did not. The authors concluded that they could predict tumour recurrence with misclassification rates of 2/34 and 3/18, respectively.

The authors presented a tree classifier with split decisions based on an a posteriori distribution over all possible splits. A description is available in the form of a technical report on the webpage of one of the authors. Due to ambiguities we did not succeed in translating the statistical ideas into software with which we could reproduce the analysis. More importantly, there is no *official* software version of the algorithm. The authors perform two dimension reduction steps before they apply the Bayesian tree classifier. First, the 12,625 probe sets on the HGU95Av2 Affymetrix GeneChips are reduced to 7,030 by excluding probe sets with a maximum intensities below 2^9 and a low variatiability across the samples. The second reduction creates 496 *metagenes* out of the 7,030 probe sets by performing k-means clustering and using the first principal component of each cluster as expression measure for the *metagene*.

As in example 1 this study is concerned with the prognosis of tumour recurrence of breast cancer patients after curative resection of the tumour. The authors state that they could not find any of the 70 genes used in the classifier of van ’t Veer et al. in any of the metagenes which come up in the Bayesian tree classifier.

The Huang example rises the following questions: Why is it impossible for us to reproduce the good classification result? Why is the classification based on the idiosyncratic method so good? The preprocessing is not part of the CV loop, which influence may this have on the misclassification rate? How is it possible to disentangle computation, technology, and biology? Can we find a link between the Huang and van ’t Veer classifiers?

3 The Compendium

Publications on microarray profiling studies often present one new microarray data set and one new classification method. Is it necessary to develop an idiosyncratic classification approach for each specific data set? Which classification result could be achieved with standard approaches? What loss in accuracy has to be traded for a rise in interpretability? How can I use new data to validate former results? If validation creates discrepancies how is it possible to assess the contribution made by algorithmic aspects: error in implementation or no success with validation? Do I have sufficient instructions and details to reproduce the method under validation in an exact way? How dependent is the classification result on the method used? What can be learned from a published profiling study for future projects? To answer these and other questions we introduce the compendium of computational diagnostic tools (CCDT)

3.1 Compendium for Computational Diagnostic Tools – CCDT

The compendium is an interactive document that calculates the misclassification rate (MCR) for different classification methods. The validation strategy is fixed but may be modified by setting specific parameters. Therefore, an outer and an inner cross-validation is mandatory: the inner CV tunes the algorithm specific parameters (including also parameters for the preprocessing) and the outer CV to estimate the misclassification rate. We see the preprocessing as part of the classification algorithm. The CV strategy is sketched in figure 1.

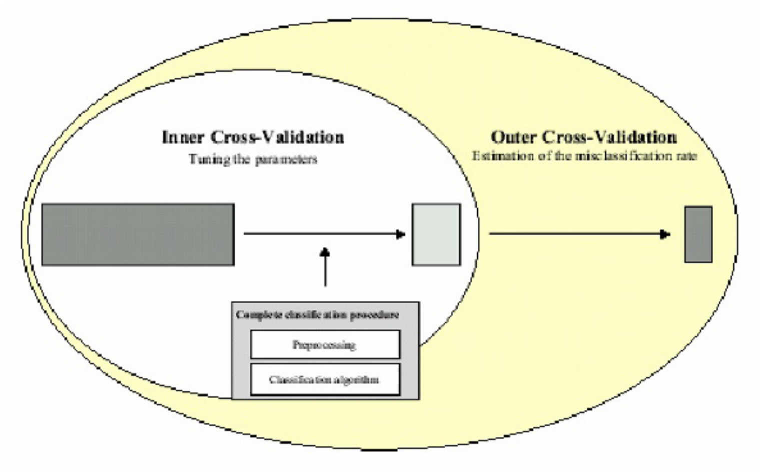


Fig. 1. The cross-validation stategy (CV - cross validation, MCR - missclassification rate)

Things that can be changed easily are: Classification methods, preprocessing steps, parameters for classification method and validation, and data set.

The compendium allows to combine guidelines with software, and to embed good statistical analysis in a text which is accessible for medical or biological researchers. It is a document that bundles primary data, processing methods (computational code), derived data, and statistical output with the textual documentation and conclusions it. Especially it contains specific tools to represent results. The inclusion of an implemented classification method is via a wrapper. Writing a wrapper function for new classification algorithms is simple. So far, five classification approaches are already implemented: Shrunken centroids (PAM) [18], support vector machines (SVM) [19], random forest [20], general partial least squares, and penalized logistic regression [21]. The compendium can be found as a package for the statistical language R at

`http://www.biometrie.uni-heidelberg.de/technical_reports.`

The compendium does not implement new algorithms but offers a framework to apply existing implementations of classification algorithms in a correct way.

3.2 Static Versus Interactive Approaches

The answer to questions like: *Which classification result could be archived with standard approaches?* or *What loss in accuracy has to be traded for a rise in interpretability?* is generally given in a paper which compares N different classification algorithms (C) and

M pre-processing (P) strategies on K different data sets (D). The N x M x K performance measures are tabulated and discussed. Dudoit et al. [12] published such a study which is extended by Lee et al. [13]. It is difficult to use their results for guidance because they certainly do not implement all algorithms of interest, the pre-processing strategies may change, and the data will become irrelevant when a new generation of microarrays is introduced. Therefore, it may be wise to replace the static by an interactive approach by offering the machinery which allows the researcher herself to perform such a study on the algorithms and pre-processing strategies of interest together with relevant data.

The compendium offers different levels of interactivity. It can be used to produce a textual output comparable with the static approach. On an intermediate level one interacts with the compendium by specifying parameters and data sets. For example, one could change the kernel of a support vector machine by simply changing the parameter `poss.pars`:

```
> poss.pars = c(list(cost = cost.range, kernel = "linear"),poss.k)
```

This is the level of sensitivity analyses or of comparing the performance of implemented algorithms on different data sets. The advanced level of interaction consists in introducing new ideas like new classification algorithms or new tools for the presentation of the results. Writing wrapper functions for new classification methods is simple. The following example shows a wrapper for diagonal discriminant analysis. The essential part is the specific definition of the predict.function by user specific needs and ideas:

```
> DLDA.wrap = function(x, y, pool = 1, ...) {
+ require(sma)
+ predict.function = function(testmatrix) {
+ res = stat.diag.da(ls = x, as.numeric(y), testmatrix,
+       pool = pool)$pred
+ return(levels(y)[res])
+ }
+ return(list(predict = predict.function, info = list()))
+ }
```

3.3 Sweave

Sweave [24] is a specific approach for generation of a dynamic report. We use Sweave as the technology for the compendium. It mixes S (R) and LATEX in a sequence of code and documentation chunks in a Rnw file. It uses S (R) for all tangling and weaving steps and hence has a very fine control over the S (R) output. Options can be set either globally to modify the default behaviour or separately for each code chunk to control how the output of the code chunks is inserted into the LATEX file.

The working principle of Sweave is sketched in figure 2 and demonstrated in the following example. The parts between `<<...>>= ...@` describe the code chunks which will be evaluated by S (R) and whose results will be woven if required into the output (again a LATEX of postscript document) or only available for later evaluation steps. The paragraphs between the code chunks will be processed as LATEX code for the output document.

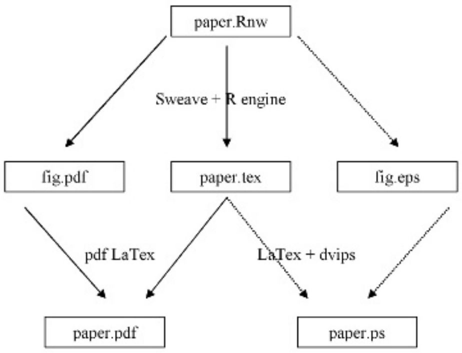


Fig. 2. The working principle of Sweave

To obtain normalized expression measures from the microarray data, Huang et al. used Affymetrix' Microarray Suite (MAS) Version 5 software.Additionally, they transformed the data to the logarithmic scale. Here, we use the function $\backslash Rfunction\{mas5\}$ in the $\backslash Rpackage\{affy\}$ library. $\backslash Robject\{eset\}$ is an object of $classexprSet$, which comprises the normalized expression values as well as the tumour sample data. All the following analyses are based on this object.

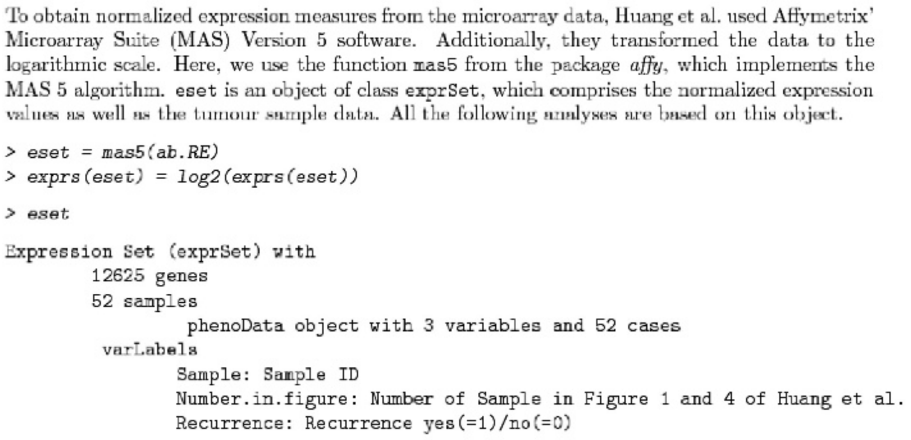
```
%%normalizing the affy batches
<<normalizing,eval=FALSE>>=
Huang.RE  <- mas5(affy.batch.RE)
exprs(Huang.RE) <- log2(exprs(Huang.RE))
@
So we have the following expression set for our further analysis
<<show1>>=
Huang.RE
@
```

The Sweave output of this part is presented in figure 3.

4 Results

The application of the compendium to data of microarray profiling study provides tools to answer crucial questions on the assessment of a new classification algorithm. A few aspects will be discussed.

A new classification algorithm can be implemented by writing the specific wrapper function which is an easy exercise. Misclassification results can be calculated and compared to the results of a set of competing algorithms. The compendium presents classification results on the basis of the confusion matrix and individual classification. Individual classification based on a specific pre-processing strategy and classification algorithm can be presented for the whole sample graphically. figure 4 shows a vote plot which presents for each subject the percentage of correct classification in the determined



To obtain normalized expression measures from the microarray data, Huang et al. used Affymetrix' Microarray Suite (MAS) Version 5 software. Additionally, they transformed the data to the logarithmic scale. Here, we use the function `mas5` from the package *affy*, which implements the MAS 5 algorithm. `eset` is an object of class `exprSet`, which comprises the normalized expression values as well as the tumour sample data. All the following analyses are based on this object.

```
> eset = mas5(ab.RE)
> exprs(eset) = log2(exprs(eset))

> eset

Expression Set (exprSet) with
        12625 genes
        52 samples
                 phenoData object with 3 variables and 52 cases
         varLabels
                Sample: Sample ID
                Number.in.figure: Number of Sample in Figure 1 and 4 of Huang et al.
                Recurrence: Recurrence yes(=1)/no(=0)
```

Fig. 3. Sweave output of example

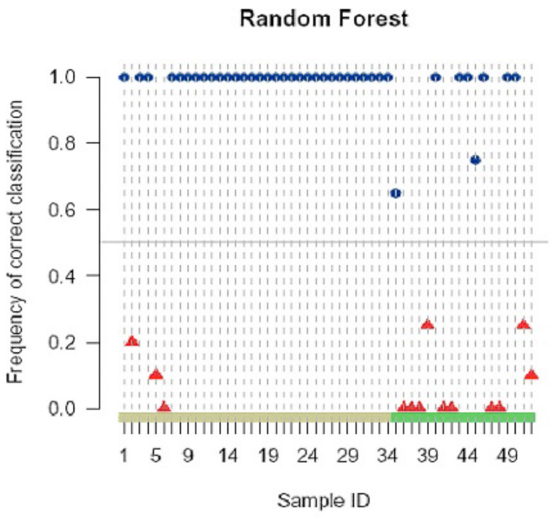
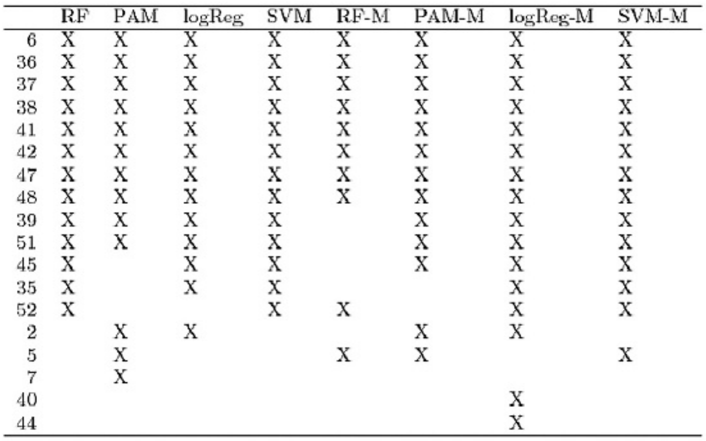


Fig. 4. Vote plot for classification result using: reanalysis of the Huang et al. data [4]

number of CV loops. The first 34 patients belong to the group with no tumour recurrence. The remaining 14 patients suffered a relapse.

A table giving a synopsis of all individual misclassifications can also be produced. Figure 5 shows all subjects who are misclassified by at least one of the classification strategies under consideration in the data of Huang et al. [4]. The individuals are ordered with respect to the number of classification strategies which lead to misclassification. Other presentations of the comparison are possible, for example a scatter plot to contrast individual MCRs between the new and the standard approaches. This idea can also be



	RF	PAM	logReg	SVM	RF-M	PAM-M	logReg-M	SVM-M
6	X	X	X	X	X	X	X	X
36	X	X	X	X	X	X	X	X
37	X	X	X	X	X	X	X	X
38	X	X	X	X	X	X	X	X
41	X	X	X	X	X	X	X	X
42	X	X	X	X	X	X	X	X
47	X	X	X	X	X	X	X	X
48	X	X	X	X	X	X	X	X
39	X	X	X	X		X	X	X
51	X	X	X	X		X	X	X
45	X		X	X		X	X	X
35	X		X	X			X	X
52	X			X	X		X	X
2		X	X			X	X	
5		X			X	X		X
7		X						
40							X	
44							X	

Fig. 5. Reanalysis of Huang et al. data (RF - random forest, PAM - shrunken centroids, logReg - penealized logistic regression, SVM - support vector machine, M - using metagenes)

implemented on the advanced level of interaction by writing the code for the respective figure.

The Huang strategy misclassifies two of the 34 patients with good prognosis and three of the 18 patients with bad prognosis. The standard algorithms give results on correct classification below 80%. Why is it impossible for us to reproduce the good classification result with standard algorithms? No implementation for the Bayesian classification tree (BCT) is available and thus no direct comparison is possible. The algorithm of the BCT is not available and its description in a technical report does not give explicit advise for its implementation into software. Therefore, we tried a sensitivity analysis by using the intermediate interaction with the compendium. First, we excluded the the preproceeing form the inner CV loop, because the original paper [4] does not take care on the adjustment of the MCR for the pre-processing strategy. This did not improve classification quality in the expected way. Second, we reduced the data set to the two-hundred most discriminating genes which introduces a huge selection bias. Only this way we could come up with 6-7 misclassifications which is still worse as the results in the original paper. Can we trust the original result? Our analysis reminds us to be quite critical to the original claims.

Huang et al. [4] state that the genes found to be crucial for the classification are different from the genes found as crucial by van ’t Veer et al. [3]. What is the reason for the missing link between the Huang and van ’t Veer classifiers? How is it possible to disentangling computation, technology, and biology? The compendium takes care on the computational part. Additionally, the BCT and the van ’t Veer classifier need to be implemented and wrapper functions for both classification algorithms have to be written. Both wrapper and the respective pre-processing strategies will be applied to both data sets. For each data set correlation between the genes used in the classifiers can be calculated and visualised by a checkerboard figure. The checkerboard for genes of both classifiers on the same data set allows to identify genes with similar expression behaviour and to study common biological aspects behind both classifiers. Comparing the checkerboards between both data sets gives hints on sample differences between both studies and discrepancies introduced by the different microarray technologies used.

5 Discussion

The literature on the induction of prognostic profiles from microarray studies is a methodological wasteland. Ambroise and McLachlan [9] describe the unthorough use of cross-validation in a number of high-profile published microarray studies. Tibshirani and Efron [7] report the difficulty in reproducing a published analysis. Huang et al. [4] present results with the potential to revolutionize clinical practice in breast cancer treatment but use an ideosyncratic statistical method which is fairly complex and neither easy to implement nor to obtain as software. A series of papers published in Nature [3], NEJM [6], [17], and The Lancet [4], [5] base their impressive results on classification methods which were developed ad-hoc for the problem at hand. The global picture looks like: the MCRs reported are of questionable clinical relevance, reanalysis of a paper does not support the strong claims they made, results are not reproducible with respect to computation, validation, or biology.

This situation has several implications: 1) It is nearly impossible to assess the value of the presented studies in terms of statistical quality and clinical impact. 2) Scientists looking for guidance to design similar studies are left puzzled by the plethora of methods. 3) It is left unclear how much potential there is for follow-up studies to incrementally improve on the results.

References

1. Microarray special. *Statistical Science*, **18**:1-117, 2003.
2. Simon R., Rademacher M.D., Dobbin K., McShane L.M.: Pitfalls in the use of DNA microarray data for diagnostic and prognostic classification. *J Nat. Cancer Inst.*, **95**: 14-18, 2003.
3. van 't Veer L., Dai H., van de Vijver M.J., He Y.D., Hart A.A.M., Mao M., Petersen H.L., van de Kooy K., Marton M.J., Witteveen A.T., Schreiber G.J., Kerkhoven R.M., Roberts C., Linsley P.S., Bernards R. and Friend S.H.: Gene expression profiling predicts clinical outcome of breast cancer. *Nature*, **415**:530-536, 2002.
4. Huang E., Cheng S.H., Dressman H., Pittman J., Tsou M.H., Horng C.F., Bild A., Iversen E.S., Liao M., Chen C.M., West M., Nevins J.R. and Huang A.T.: Gene expression predictors of breast cancer outcomes. *The Lancet*, **361**:1590-1596, 2003.
5. Chang J., Wooten E., Tsimelzon A., Hilsenbeck S., Gutierrez C, Elledge R., Mohsin S., Osborne K., Chamness G., Allred C., O'Connell P.: Gene expression profiling for the prediction of therapeutic response to docetaxel in patients with breast cancer. *The Lancet*, **362**:362-369, 2003.
6. Bullinger L., Döhner K., Bair E., Fröhling S., Schlenk R.F., Tibshirani R., Döhner H., Pollack J.R.: Use of Gene-Expression Profiling to Identify Prognostic Subclasses in Adult Acute Myeloid Leukemia. *NEJM*, **350**:1605-1616, 2004.
7. Tibshirani R.J., Efron B.: Pre-validation and inference in microarrays. *Statistical Applications in Genetics and Molecular Biology*, **1**:1, 2002.

8. Breiman L.: Statistical Modelling: The Two Cultures. *Statistical Science*, **16**:199-231, 2001.
9. Ambroise C., McLachlan G.J.: Selection bias in gene extraction on the basis of microarray gene-expression data. *Proc Natl Acad Sci*, **99**:6562-6566, 2002.
10. Brenton J.D., Caldas C.: Predictive cancer genomics - what do we need? *The Lancet*, **362**:340-341, 2003.
11. Leisch F., Rossini A.J.: Reproducible statistical research. *Chance*, **16**:41-46, 2003.
12. Dudoit S., Fridlyand J., Speed T.P.: Comparison of Discrimination Methods for the Classification of Tumors Using Gene Expression Data. *Journal of the American Statistical Association*, **97**:77-87, 2002.
13. Lee J.W., Korea University, Department of statistics, personal communication.
14. Ihaka R., Gentleman R.: R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, **5**:299-314, 1996.
15. Gentleman R., Carey V.: Bioconductor. *R News*, **2(1)**:11-16, 2002.
16. Leisch F.: Dynamic generation of statistical reports using literate data analysis. *Compstat 2002 - Proceedings in Computational Statistics*, 575-580, 2002.
17. van de Vijver M.J., He Y.D., van 't Veer L.J. et al.: A gene-expression signature as a predictor of survival in breast cancer. *N Engl J Med*, **347**:1999-2009, 2002.
18. Tibshirani R., Hastie T., Narasimhan B., Chu G.: Class prediction by nearest shrunken centroids, with application to DNA microarrays. *Statistical Science*, **18**:104-117, 2003.
19. Vapnik V.: *The Nature of Statistical Learning Theory*. Springer, New York (1999)
20. Breiman L.: Random Forests. *Machine Learning Journal*, **45**:5-32, 2001.
21. Eilers P.H., Boer J.M., Van Ommen G.J., Van Houwelingen H.C.: Classification of Microarray Data with Penalized Logistic Regression. *Proceedings of SPIE volume 4266:progress in biomedical optics and imaging*, **2**:187-198, 2001.
22. Carey V.J.: Literate Statistical Programming: Concepts and Tools. *Chance*, **14**:46-50, 2001.
23. Sawitzki, G.: Keeping Statistics Alive in Documents. *Computational Statistics*, **17**:65-88, 2002.
24. Leisch, F.: Dynamic generation of statistical reports using literate data analysis. *Compstat 2002 - Proceedings in Computational Statistics*, 575-580, 2002.

Step-Down FDR Procedures for Large Numbers of Hypotheses

Paul N. Somerville

Emeritus Professor of Statistics
University of Central Florida
2805 N. Hwy. A1A, Unit 502
Indialantic, FL 32903
somervil@pegasus.cc.ucf.edu

Abstract. Somerville (2004b) developed FDR step-down procedures which were particularly appropriate for cases where the number of false hypotheses was small. The test statistics were assumed to have a multivariate-t distribution with common correlation. MCV's (minimum critical values) were chosen so that 8 unique critical values resulted. Tables were given for numbers of hypotheses m, ranging from 50 to 10,000, for $\rho = 0., 0.5$, and $\nu = 15, \infty$. In this paper we extend the results, using MCV's resulting in 31 critical values. Tables are given for the same values of m, for $\rho = 0, 0.1, 0.5$ and $\nu = 15, \infty$. Interpolation rules are given for m, ρ and ν. Use of larger numbers of critical values increase both the power and the number of hypotheses falsely rejected. When the expected number of false hypotheses is small, use of the procedures of this paper results in a reduced number of false rejections with a negligible reduction in power.

1 Introduction

There are many situations where a researcher is interested in the outcome of a family of tests. An example is the case where m experimental drugs are compared to a standard with respect to an outcome. Tests can be made of the m null hypotheses of no effect, each at a level α. This would be testing at a per comparison error rate (PCER) of α. For $m > 1$, the probability of rejecting some null hypothesis when it was true would be larger than α. To protect from such situations arising from multiplicity, a standard procedure has been to use a family-wise error rate (FWER). In this case, the family-wise error rate would be the probability of rejecting at least one null hypothesis when in fact they are all true. A common single-step method to achieve the family-wise error rate is to use the Bonferroni procedure, discussed by R. A. Fisher (1935). Other single step procedures include those of Scheffe (1953) for testing several linear combinations, Tukey (1953) for pair-wise comparisons and Dunnett (1955) for comparisons with a control.

Multi-step (step-wise) FWER procedures have been developed which are more powerful than single-step procedures. One of the first was introduced by Naik (1975). Others include step-up and step-down procedures developed for comparisons with a control by Dunnett and Tamhane (1991, 1992).

More recently multi-step procedures which control the “false discovery rate" FDR (expected proportion of “false discoveries" or Type I errors) say q have been proposed.

One motivation was the under-utilization of FWER procedures because of their perceived lack of “power". Benjamini and Hochberg (1995) presented a step-up procedure, valid for independent test statistics. Benjamini and Liu (1999a) presented a step-down procedure valid under the same conditions. Troendle (2000) developed both step-up and step-down procedures which asymptotically control the FDR when the test statistics have a multivariate-t distribution. Benjamini and Liu (1999b) and Benjamini and Yekutieli (1999) gave distribution free FDR procedures. Benjamini and Yekutieli (2001) showed that the procedure of Benjamini and Hochberg (1995) was also valid when the p-values were “positively dependent". Benjamini, Krieger and Yekutieli (2001) developed a two stage step-up procedure. Sarkar (2002) has made important contributions.

Somerville (2004b), using least favorable configurations, developed both step-up and step-down procedures which are valid for dependent or independent hypotheses. All calculations assume the test statistics have a joint multivariate-t distribution and a common correlation coefficient ρ. The concept of “Minimum Critical Value" (MCV) was introduced. When the MCV (see section 2.) is equal to 0 the step-down procedure is the same as that of Troendle (2000). Studies comparing powers of various procedures by Horne and Dunnett (2003) and Somerville (2003) have shown the methods of Troendle (2000) and Somerville (2004b) to be among the most powerful.

FDR procedures, while controlling the expected number of “false positives", have also been criticized for not controlling the actual number or proportion. van der Laan, Dudoit and Pollard (2004) generalize FWER procedures, introducing gFWER which controls u, the number of Type I errors. They also introduce PFP to control γ, the proportion of “false positives".

In this paper we introduce step-down FDR procedures which use large MCV’s (Minimum Critical Values). These procedures not only control the expected FDR, but also result in values of $P[u \leq 2]$ which compare with those achieved by Korn et al and van der Laan et al when the number of false hypotheses is small, or where finding a small number of false hypotheses is a satisfactory outcome. Six tables of critical values d_{m-30} to d_m are given for $50 \leq m \leq 10,000$. Instead of using $MCV = d_{m-30}$, and 31 “unique" critical values, the tables may be “truncated" by elimination of the smallest values. Reducing the number of “unique" critical values, reduces the probability of “a or fewer" false positives, where the value of a is arbitrary.

2 Minimum Critical Values

Assuming a multivariate-t distribution of the test statistics with a common correlation coefficient ρ, Somerville (2003) observed that, in most situations, the smallest of the calculated FDR critical values was less than the commonly used critical value of $t_{\alpha,\nu}$ for comparing two means, where $t_{\alpha,\nu}$ is the value of student’s t with type I error of α and ν degrees of freedom. In many situations it was negative, and when the number of tests was large, one or more of the critical values could be negatively infinite. This situation would certainly be unappealing to most users, and motivated the concept of “minimum critical value" (MCV). The “minimum critical value" is defined to be the smallest critical value, which, when exceeded or equaled, would result in an hypothesis rejection. The “minimum critical value" may be more or less arbitrarily chosen. If the calculated value for d_i is smaller than the chosen MCV, it is replaced by the MCV value, and d_{i+1}, d_{i+2}, are sequentially calculated, where $d_i \leq d_j$ when $i \leq j$.

Somerville (2003, 2004a,b) made many calculations finding convincing empirical evidence, especially for large values of m, that if n_F were the actual number of false hypotheses, that there was an inverse relationship between n_F and the value of the MCV which maximized the "power" of the FDR procedure. That is, if n_F were large, the MCV for maximum power of the procedure should be small and for small values of n_F , the MCV should be large. In particular, Somerville observed that, for the values of m studied, choosing the value of MCV such that the number of "unique" critical values was equal to n_F resulted in "powers" near the maximum. An approximately equivalent result could be obtained by automatically "truncating" the step-down FDR process after n_F steps. It may be noted (Somerville (2003, 2004b)) that if the MCV is equal to the corresponding critical value for the single-step test, all the critical values of the FDR procedure are equal.

3 Tables for the FDR Step-Down Procedure when the Number of False Hypotheses Is Small

Tables 2, 3 and 4 give step-down FDR critical values for m ranging from 50 to 10,000 when $\nu = 15$ or $\infty, q = .05$ for the procedure, and ρ = 0, .1 and .5 respectively, when the hypotheses are one-sided. For each combination of m, ρ and ν. MCV was chosen as the smallest value such that $d_1 = d_2 = \ldots = d_{m-30} = MCV$ results in the FDR less than or equal to q. There are thus, for each m in each table, exactly 31 "unique" critical values, a more or less arbitrarily chosen "small" value.

There may be error in the third decimal place in the tables. Critical values for m not included in the tables can be obtained by interpolation (or extrapolation if $m < 50,000$ (say)). Each critical value d_i in a table is approximately linear in $ln(m)$. Critical values for $15 < \nu < \infty$ can be obtained by linear interpolation in $1/\nu$. Critical values for $0 < \rho < .5$, can be approximated using quadratic interpolation. It is worth noting that as $\rho \rightarrow 1$, all critical values are equal to z_q (or t_q), the value of the normal (or t) variate which is exceeded q of the time.

4 Calculation of Critical Values and Powers

Fortran 90 programs SEQDN and SEQUP can sequentially calculate the critical values d_2 to d_m for step-down and step-up FDR, respectively, for arbitrary values of m, q, ρ, and ν. N ($10^5, 10^6$ or 10^7) random normal multivariate vectors of size m were used to obtain each critical value.

Fortran 90 programs FDRPWRDN and FDRPWRUP calculate powers, E(Q), $P[u \leq 1, 2, \ldots, 7]$, and $P[\gamma \leq .05, .10, .15]$ where u and γ are the number and proportion of false discoveries respectively. Inputs are $m, \rho, \nu, \Delta, n_F$ and a set of m critical values. Δ is the common standardized mean of the test statistics corresponding to the n_F false hypotheses. N random normal multivariate vectors of size m are used. Three kinds of power are always calculated: per pair, all pairs and any pair (see Horn and Dunnett

(2004)), and also E(Q). The probability of rejecting at least one of the false hypotheses is called any pairs power. The probability of rejecting all false hypotheses is called all pairs power. Considering a specific hypothesis, the probability of its rejection is called the per pairs power. Since our calculations assume all the test statistics corresponding to the F hypotheses have the same location parameter, the per pair power is identical to the average power.

5 Example

Korn, Troendle, McShane and Simon (2003) recently proposed two new procedures which control, with specified confidence, the actual number of false discoveries, and the actual proportion of false discoveries, respectively. They applied their procedures to analyze a microarray dataset consisting of measurements on approximately 9000 genes in paired tumor specimens, collected both before and after chemotherapy on 20 breast cancer patients. Their study, after elimination of cases of missing data included 8029 genes for analysis. The object was to simultaneously test the null hypotheses that the mean pre and post chemotherapy expression of genes was the same. Their Procedure A identified 28 genes where u, the number of false discoveries, was ≤ 2, with confidence .05. Procedure B identified the same 28 genes where γ, the false discovery proportion, was $\leq .10$ with approximate confidence .95.

Table 1. Table A: Number of genes identified by procedures, $(m = 8029, q = 0.05)$. Table B: Minimum values of $P[u \leq 2]$ $(m = 8029, q = .05, \nu = 3.464)$. Rough approximate value of n_F at minimum in parentheses

Table A

	$\nu = \infty$			$\nu = 15$
MCV	$\rho = .0$	$\rho = .1$	$\rho = .5$	$\rho = .0$
d_{m-7}	20	21	29	24
d_{m-11}	23	23	33	27
d_{m-15}	24	24	35	29
d_{m-19}	27	27	38	31
d_{m-23}	28	29	40	33
d_{m-27}	29	29	> 50	33
d_{m-30}	29	33	> 50	36

Table B

	$\nu = \infty$			$\nu = 15$
MCV	$\rho = .0$	$\rho = .1$	$\rho = .5$	$\rho = .0$
d_{m-7}	.99(40)	.96(70)	.95(50)	.90(50)
d_{m-11}	.98(50)	.94(70)	.93(55)	.90(50)
d_{m-15}	.96(70)	.91(70)	.92(55)	.89(80)
d_{m-19}	.92(80)	.88(70)	.91(60)	.89(80)
d_{m-23}	.88(80)	.85(70)	.91(60)	.88(90)
d_{m-27}	.83(80)	.82(80)	.90(60)	.88(90)
d_{m-30}	.78(90)	.78(90)	.87(70)	.88(100)

The tables 2, 3 and 4 were used to test the same hypotheses, "truncating" each table to utilize exactly 8, 12, 16, 20, 24, 28 and 31 "unique" critical values ("truncating" to exactly 12 "unique" critical values would mean setting MCV equal to d_{m-11}, and setting $d_i = MCV$ for $i < m - 11$.). Assuming 15 degrees of freedom for each of the 8029 tests, the corresponding FDR step-down critical values were obtained using the Fortran program SEQDN. Table 1A indicates the number of genes identified by the procedures.

Table 1B illustrates the control of the number of false positives, respectively for $m = 8029$. The graph $P[u \leq 2]$ vs. n_F is (shallow) bowl shaped. The value in parentheses is a very rough estimate of the value of n_F for the minimum.

Table 2. Step-down FDR Critical Values (31 “unique") for $\rho = 0$ and $q = 0.05$ (the first columns should be read as d_i where $i = m, m-1, \ldots, m-30, \ldots, 1$)

$Table\, C(\rho = 0, \nu = 15, q = 0.05)$

m	50	100	250	500	1000	2500	5000	10000
m	3.632	3.928	4.306	4.579	4.842	5.177	5.420	5.656
$m-1$	3.332	3.647	4.045	4.332	4.614	4.962	5.214	5.464
$m-2$	3.140	3.467	3.876	4.173	4.458	4.822	5.081	5.340
$m-3$	2.999	3.335	3.753	4.054	4.345	4.710	4.976	5.225
$m-4$	2.885	3.231	3.656	3.961	4.254	4.630	4.899	5.150
$m-5$	2.789	3.141	3.573	3.884	4.182	4.560	4.827	5.100
$m-6$	2.705	3.065	3.503	3.816	4.066	4.444	4.723	4.979
$m-7$	2.630	2.999	3.443	3.760	4.066	4.444	4.723	4.979
$m-8$	2.561	2.937	3.389	3.705	4.013	4.392	4.672	4.941
$m-9$	2.499	2.885	3.338	3.660	3.967	4.357	4.631	4.912
$m-10$	2.441	2.835	3.293	3.617	3.929	4.306	4.585	4.847
$m-11$	2.383	2.787	3.249	3.577	3.886	4.283	4.567	4.834
$m-12$	2.333	2.743	3.213	3.543	3.855	4.244	4.527	4.803
$m-13$	2.282	2.702	3.176	3.504	3.824	4.220	4.488	4.774
$m-14$	2.233	2.666	3.144	3.474	3.784	4.182	4.459	4.746
$m-15$	2.187	2.627	3.111	3.446	3.759	3.153	4.441	4.735
$m-16$	2.141	2.591	3.081	3.416	3.731	4.124	4.426	4.677
$m-17$	2.079	2.559	3.051	3.388	3.704	4.113	4.394	4.677
$m-18$	2.052	2.526	3.025	3.362	3.679	4.078	4.359	4.645
$m-19$	2.010	2.495	3.000	3.338	3.669	4.063	4.346	4.625
$m-20$	1.969	2.464	3.973	3.315	3.636	4.025	4.329	4.602
$m-21$	1.926	2.436	2.949	3.292	3.605	4.024	4.303	4.577
$m-22$	1.884	2.406	2.926	3.272	3.592	3.994	4.279	4.567
$m-23$	1.842	2.380	2.905	3.250	3.570	3.980	4.249	4.532
$m-24$	1.801	2.352	2.883	3.227	3.554	3.952	4.238	4.532
$m-25$	1.759	2.328	2.861	3.207	3.538	3.937	4.221	4.507
$m-26$	1.716	2.302	2.839	3.192	3.513	3.925	4.203	4.494
$m-27$	1.670	2.275	2.822	3.169	3.493	3.890	4.185	4.457
$m-28$	1.625	2.249	2.797	3.152	3.465	3.875	4.185	4.445
$m-29$	1.625	2.243	2.790	3.138	3.454	3.842	4.136	4.408
$m-30$	1.464	2.152	2.712	3.062	3.385	3.784	4.065	4.343
.	.	.	.	.	.	.	.	
1	1.464	2.152	2.712	3.062	3.385	3.784	4.065	4.343
$ln(m)$	3.912	4.605	5.521	6.215	6.908	7.824	8.517	9.210

$Table\, D(\rho = 0, \nu = \infty, q = 0.05)$

m	50	100	250	500	1000	2500	5000	10000
m	3.083	3.283	3.533	3.713	3.884	4.102	4.259	4.412
$m-1$	2.867	3.081	3.346	3.537	3.714	3.938	4.103	4.266
$m-2$	2.731	2.958	3.234	3.426	3.613	3.842	4.009	4.172
$m-3$	2.629	2.865	3.150	3.349	3.538	3.772	3.940	4.106
$m-4$	2.545	2.791	3.084	3.286	3.477	3.713	3.887	4.054
$m-5$	2.474	2.729	3.027	3.234	3.428	3.672	3.842	4.011
$m-6$	2.412	2.676	2.980	3.190	3.387	3.635	3.804	3.974
$m-7$	2.354	2.626	2.938	3.150	3.350	3.596	3.771	3.941
$m-8$	2.302	2.585	2.899	3.116	3.317	3.566	3.742	3.915
$m-9$	2.254	2.543	2.865	3.084	3.286	3.543	3.717	3.887
$m-10$	2.207	2.507	2.835	3.054	3.260	3.511	3.694	3.865
$m-11$	2.165	2.470	2.806	3.029	3.235	3.488	3.669	3.842
$m-12$	2.122	2.440	2.777	3.003	3.213	3.468	3.655	3.824
$m-13$	2.082	2.407	2.753	2.980	3.190	3.448	3.633	3.803
$m-14$	2.044	2.379	2.730	2.960	3.169	3.428	3.613	3.788
$m-15$	2.006	2.352	2.705	2.937	3.151	3.411	3.602	3.773
$m-16$	1.969	2.324	2.684	2.919	3.133	3.399	3.582	3.757
$m-17$	1.932	2.300	2.666	2.901	3.117	3.376	3.566	3.743
$m-18$	1.897	2.274	2.644	2.881	3.099	3.365	3.557	3.728
$m-19$	1.861	2.251	2.625	2.867	3.084	3.354	3.537	3.719
$m-20$	1.826	2.223	2.609	2.849	3.069	3.334	3.526	3.704
$m-21$	1.791	2.206	2.590	2.833	3.055	3.323	3.516	3.693
$m-22$	1.755	2.181	2.573	2.821	3.041	3.308	3.502	3.685
$m-23$	1.720	2.160	2.557	2.804	3.029	3.308	3.484	3.672
$m-24$	1.685	2.138	2.541	2.792	3.016	3.282	3.484	3.660
$m-25$	1.648	2.121	2.531	2.780	3.003	3.275	3.469	3.654
$m-26$	1.621	2.098	2.509	2.764	2.994	3.269	3.455	3.644
$m-27$	1.549	2.066	2.490	2.746	2.976	3.249	3.443	3.628
$m-28$	1.549	2.066	2.490	2.746	2.969	3.249	3.443	3.628
$m-29$	1.549	2.066	2.490	2.746	2.969	3.249	3.443	3.628
$m-30$	1.397	1.991	2.437	2.700	2.935	3.212	3.408	3.594
.	.	.	.	.	.	.	.	.
1	1.397	1.991	2.437	2.700	2.935	3.212	3.408	3.594
$ln(m)$	3.912	4.605	5.521	6.215	6.908	7.824	8.517	9.210

6 Summary and Conclusion

Step-down FDR procedures have been developed for the case where there are many hypotheses. The procedures are particularly appropriate when relatively few hypotheses are false, or where obtaining a limited number of “discoveries" is satisfactory. For $50 \leq m \leq 10,000$ and q = .05, tables of critical values of the test statistics are presented for $\rho = 0$, .1 and .5 when $\nu = \infty$, or $\nu = 15$. There are 31 “unique" tabulated critical values, but the tables may be “truncated". Setting MCV equal to d_m results in a single unique critical value and a single step procedure. This choice yields the largest $P[u \leq a]$ where a is arbitrary. Choosing the appropriate value for MCV is an attempt to balance the conflicting goals of rejecting all false hypotheses and not rejecting those hypotheses which are true. Empirical results suggest that “powers" are maximized when the MCV is chosen so that the number of “unique" critical values is equal to n_F (of course almost always unknown).

Some additional observations and conclusions are:

1. i) $P[u \leq a]$ decreases as the number of unique critical values used increases. Thus the number of critical values can be chosen so as to control $P[u \leq a]$.

Table 3. Step-down FDR Critical Values (31 “unique") for $\rho = 0.1$ and $q = 0.05$ (the first columns should be read as d_i where $i = m, m-1, \ldots, m-30, \ldots, 1$)

Table E$(\rho = 0.1, \nu = 15, q = 0.05)$

m	50	100	250	500	1000	2500	5000	10000
m	3.580	3.860	4.217	4.474	4.719	5.033	5.260	5.479
$m-1$	3.295	3.597	3.975	4.245	4.505	4.831	5.074	5.296
$m-2$	3.113	3.427	3.816	4.097	4.364	4.703	4.947	5.171
$m-3$	2.987	3.300	3.701	3.985	4.261	4.597	4.848	5.092
$m-4$	2.867	3.201	3.608	3.899	4.176	4.536	4.777	5.018
$m-5$	2.774	3.115	3.530	3.825	4.107	4.459	4.711	4.957
$m-6$	2.693	3.041	3.465	3.764	4.050	4.406	4.662	4.900
$m-7$	2.619	2.980	3.405	3.706	3.997	4.355	4.597	4.853
$m-8$	2.552	2.918	3.356	3.658	3.934	4.302	4.581	4.822
$m-9$	2.491	2.868	3.306	3.612	3.906	4.276	4.522	4.778
$m-10$	2.434	2.820	3.264	3.572	3.851	4.223	4.507	4.742
$m-11$	2.379	2.773	3.220	3.535	3.834	4.202	4.451	4.722
$m-12$	2.328	2.730	3.186	3.502	3.796	4.159	4.428	4.691
$m-13$	2.279	2.691	3.150	3.466	3.759	4.144	4.395	4.650
$m-14$	2.231	2.655	3.119	3.434	3.742	4.102	4.395	4.623
$m-15$	2.186	2.617	3.086	3.406	3.708	4.066	4.350	4.623
$m-16$	2.141	2.583	3.058	3.380	3.679	4.054	4.333	4.573
$m-17$	2.098	2.549	3.028	3.350	3.648	4.054	4.299	4.556
$m-18$	2.054	2.519	3.003	3.330	3.641	4.001	4.281	4.545
$m-19$	2.010	2.486	2.980	3.307	3.601	3.986	4.253	4.545
$m-20$	1.973	2.456	2.952	3.278	3.601	3.958	4.237	4.495
$m-21$	1.929	2.432	2.929	3.260	3.563	3.958	4.213	4.484
$m-22$	1.888	2.400	2.908	3.238	3.538	3.919	4.213	4.440
$m-23$	1.847	2.375	2.884	3.218	3.527	3.913	4.188	4.439
$m-24$	1.807	2.346	2.863	3.195	3.501	3.889	4.162	4.419
$m-25$	1.765	2.324	2.841	3.176	3.494	3.865	4.136	4.419
$m-26$	1.720	2.297	2.820	3.156	3.457	3.853	4.124	4.396
$m-27$	1.682	2.269	2.801	3.135	3.439	3.833	4.101	4.355
$m-28$	1.634	2.244	2.776	3.116	3.425	3.806	4.094	4.350
$m-29$	1.622	2.229	2.760	3.099	3.425	3.780	4.046	4.307
$m-30$	1.453	2.133	2.681	3.020	3.325	3.708	3.978	4.240
.	.	.	.	.	.	.	.	.
1	1.453	2.133	2.681	3.020	3.325	3.708	3.978	4.240
$ln(m)$	3.912	4.605	5.521	6.215	6.908	7.824	8.517	9.210

Table F$(\rho = 0.1, \nu = \infty, q = 0.05)$

m	50	100	250	500	1000	2500	5000	10000
m	3.070	3.268	3.516	3.693	3.864	4.081	4.235	4.386
$m-1$	2.858	3.072	3.335	3.522	3.700	3.924	4.086	4.242
$m-2$	2.724	2.949	3.224	3.416	3.599	3.829	3.995	4.153
$m-3$	2.623	2.859	3.141	3.339	3.526	3.759	3.929	4.091
$m-4$	2.542	2.786	3.076	3.277	3.468	3.704	3.880	4.041
$m-5$	2.471	2.725	3.021	3.226	3.420	3.660	3.832	3.999
$m-6$	2.409	2.670	2.974	3.181	3.377	3.621	3.795	3.963
$m-7$	2.352	2.623	2.932	3.142	3.343	3.588	3.763	3.932
$m-8$	2.300	2.580	2.895	3.107	3.309	3.558	3.735	3.907
$m-9$	2.253	2.541	2.862	3.077	3.280	3.531	3.708	3.876
$m-10$	2.207	2.503	2.829	3.049	3.255	3.503	3.685	3.858
$m-11$	2.164	2.469	2.801	3.023	3.227	3.481	3.662	3.834
$m-12$	2.122	2.437	2.774	2.998	3.205	3.461	3.643	3.817
$m-13$	2.084	2.407	2.749	2.975	3.185	3.442	3.625	3.797
$m-14$	2.045	2.377	2.725	2.954	3.161	3.424	3.609	3.778
$m-15$	2.008	2.350	2.703	2.934	3.147	3.403	3.586	3.766
$m-16$	1.970	2.324	2.681	2.914	3.127	3.389	3.572	3.749
$m-17$	1.937	2.298	2.661	2.896	3.110	3.372	3.560	3.735
$m-18$	1.899	2.274	2.641	2.878	3.096	3.359	3.542	3.720
$m-19$	1.866	2.250	2.623	2.862	3.078	3.343	3.533	3.710
$m-20$	1.830	2.226	2.605	2.845	3.064	3.331	3.513	3.698
$m-21$	1.795	2.204	2.587	2.831	3.052	3.313	3.506	3.680
$m-22$	1.760	2.182	2.572	2.813	3.033	3.309	3.491	3.672
$m-23$	1.727	2.161	2.555	2.803	3.025	3.289	3.488	3.663
$m-24$	1.691	2.138	2.539	2.786	3.008	3.278	3.464	3.652
$m-25$	1.654	2.121	2.523	2.773	2.997	3.269	3.459	3.639
$m-26$	1.622	2.099	2.510	2.760	2.985	3.259	3.449	3.630
$m-27$	1.574	2.074	2.492	2.747	2.969	3.243	3.439	3.623
$m-28$	1.544	2.059	2.480	2.734	2.969	3.239	3.426	3.607
$m-29$	1.544	2.059	2.480	2.733	2.956	3.233	3.424	3.593
$m-30$	1.388	1.978	2.420	2.681	2.912	3.187	3.381	3.568
.	.	.	.	.	.	.	.	.
1	1.388	1.978	2.420	2.681	2.912	3.187	3.381	3.568
$ln(m)$	3.912	4.605	5.521	6.215	6.908	7.824	8.517	9.210

2. ii) The number of hypotheses rejected increases with the number of “unique" critical values used in the step-down procedure. The probability of false rejection also increases.
3. iii) For sufficiently large n_F, as n_F increases $P[u \leq 2]$ increases and approaches 1.
4. iv) The tables have been calculated under the assumption that the test statistics have a joint multivariate-t distribution with common correlation coefficient ρ. However, the critical values tabulated for the test statistic can be converted to critical p-values. These converted p-values may be useful for cases where the joint distribution of the test statistics is other than multivariate-t or that any individual test is one- two sided.
5. v) Use of the step-down FDR procedure of this paper is an alternative to procedure A of Korn et al and the generalized procedure of van der Laan, Dudoit and Pollard.
6. vi) Underestimating ρ results in the identification of fewer false hypotheses.
7. vii) Use of tables 2, 3 and 4 can make using step-down FDR procedures simple and easily manageable, though requiring some normality and correlation assumptions. The procedures control the FDR, and the number of false positives can be controlled by “truncation" choice. Critical values for parameters not given by the tables can be obtained using the Fortran program SEQDN.

Table 4. Step-down FDR Critical Values (31 "unique") for $\rho = 0.5$ and $q = 0.05$ (the first columns should be read as d_i where $i = m, m-1, \ldots, m-30, \ldots, 1$)

$Table\, G(\rho = 0.5, \nu = 15, q = 0.05)$

m	50	100	250	500	1000	2500	5000	10000
m	3.243	3.452	3.711	3.896	4.067	4.294	4.455	4.608
$m-1$	3.039	3.266	3.545	3.738	3.929	4.155	4.322	4.483
$m-2$	2.902	3.144	3.432	3.634	3.826	4.057	4.242	4.409
$m-3$	2.798	3.048	3.349	3.558	3.750	3.990	4.164	4.328
$m-4$	2.713	2.974	3.284	3.497	3.693	3.943	4.118	4.303
$m-5$	2.641	2.909	3.223	3.444	3.648	3.905	4.076	4.255
$m-6$	2.575	2.853	3.179	3.397	3.607	3.865	4.038	4.191
$m-7$	2.514	2.805	3.129	3.358	3.563	3.812	4.004	4.161
$m-8$	2.460	2.755	3.096	3.314	3.528	3.786	3.972	4.133
$m-9$	2.410	2.717	3.056	3.289	3.494	3.758	3.944	4.106
$m-10$	2.362	2.678	3.026	3.256	3.468	3.732	3.918	4.080
$m-11$	2.316	2.642	2.989	3.233	3.444	3.709	3.894	4.056
$m-12$	2.274	2.606	2.963	3.199	3.420	3.690	3.873	4.033
$m-13$	2.232	2.571	2.937	3.181	3.397	3.670	3.853	4.011
$m-14$	2.189	2.543	2.913	3.145	3.375	3.642	3.835	3.991
$m-15$	2.152	2.514	2.878	3.131	3.354	3.623	3.819	3.972
$m-16$	2.111	2.478	2.864	3.103	3.333	3.605	3.803	3.954
$m-17$	2.077	2.452	2.836	3.095	3.313	3.588	3.789	3.938
$m-18$	2.058	2.428	2.815	3.064	3.293	3.571	3.777	3.923
$m-19$	1.995	2.395	2.797	3.045	3.274	3.555	3.760	3.909
$m-20$	1.967	2.369	2.769	3.026	3.256	3.539	3.749	3.896
$m-21$	1.924	2.352	2.755	3.019	3.238	3.523	3.729	3.884
$m-22$	1.884	2.323	2.738	2.991	3.221	3.507	3.717	3.873
$m-23$	1.849	2.296	2.710	2.973	3.205	3.491	3.695	3.842
$m-24$	1.811	2.269	2.695	2.957	3.190	3.474	3.676	3.816
$m-25$	1.770	2.250	2.671	2.936	3.175	3.458	3.659	3.794
$m-26$	1.726	2.221	2.655	2.915	3.160	3.440	3.644	3.776
$m-27$	1.685	2.192	2.634	2.901	3.145	3.421	3.626	3.763
$m-28$	1.639	2.162	2.606	2.872	3.130	3.401	3.611	3.754
$m-29$	1.585	2.132	2.578	2.857	3.115	3.380	3.573	3.750
$m-30$	1.396	2.015	2.486	2.762	3.003	3.305	3.499	3.689
.	.	.	.	.	.	.	.	
1	1.396	2.015	2.486	2.762	3.003	3.305	3.499	3.689
$ln(m)$	3.912	4.605	5.521	6.215	6.908	7.824	8.517	9.210

$Table\, H(\rho = 0.5, \nu = \infty, q = 0.05)$

m	50	100	250	500	1000	2500	5000	10000
m	2.883	3.047	3.252	3.393	3.528	3.696	3.816	3.943
$m-1$	2.714	2.894	3.113	3.272	3.416	3.599	3.717	3.848
$m-2$	2.605	2.798	3.025	3.183	3.332	3.522	3.647	3.777
$m-3$	2.522	2.724	2.962	3.119	3.272	3.463	3.599	3.725
$m-4$	2.454	2.665	2.905	3.073	3.228	3.418	3.558	3.686
$m-5$	2.394	2.612	2.866	3.036	3.189	3.383	3.527	3.657
$m-6$	2.340	2.566	2.823	3.003	3.155	3.356	3.497	3.629
$m-7$	2.291	2.525	2.792	2.967	3.127	3.332	3.472	3.606
$m-8$	2.249	2.489	2.759	2.938	3.103	3.306	3.448	3.584
$m-9$	2.205	2.456	2.729	2.912	3.080	3.283	3.426	3.564
$m-10$	2.164	2.424	2.704	2.887	3.058	3.261	3.406	3.546
$m-11$	2.129	2.394	2.676	2.864	3.037	3.241	3.388	3.528
$m-12$	2.093	2.367	2.655	2.843	3.017	3.223	3.370	3.512
$m-13$	2.059	2.338	2.635	2.823	2.999	3.206	3.354	3.497
$m-14$	2.022	2.313	2.610	2.804	2.981	3.190	3.339	3.483
$m-15$	1.990	2.290	2.594	2.786	2.964	3.175	3.325	3.469
$m-16$	1.959	2.262	2.576	2.769	2.948	3.162	3.312	3.457
$m-17$	1.925	2.243	2.553	2.753	2.932	3.149	3.300	3.445
$m-18$	1.891	2.218	2.539	2.738	2.917	3.136	3.288	3.433
$m-19$	1.864	2.205	2.523	2.723	2.903	3.124	3.276	3.422
$m-20$	1.831	2.175	2.499	2.708	2.889	3.113	3.265	3.411
$m-21$	1.800	2.154	2.491	2.694	2.876	3.101	3.254	3.400
$m-22$	1.767	2.136	2.469	2.679	2.863	3.090	3.243	3.390
$m-23$	1.734	2.113	2.460	2.665	2.851	3.078	3.232	3.379
$m-24$	1.697	2.092	2.442	2.650	2.838	3.066	3.221	3.368
$m-25$	1.670	2.071	2.419	2.635	2.826	3.054	3.209	3.357
$m-26$	1.630	2.053	2.411	2.620	2.814	3.041	3.197	3.346
$m-27$	1.594	2.029	2.390	2.603	2.803	3.027	3.184	3.334
$m-28$	1.554	2.004	2.375	2.586	2.782	3.013	3.170	3.322
$m-29$	1.512	1.997	2.356	2.569	2.767	2.997	3.154	3.308
$m-30$	1.335	1.879	2.278	2.505	2.703	2.936	3.096	3.245
.	.	.	.	.	.	.	.	.
1	1.335	1.879	2.278	2.505	2.703	2.936	3.096	3.245
$ln(m)$	3.912	4.605	5.521	6.215	6.908	7.824	8.517	9.210

References

1. Benjamini Y., Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. J.R. Statist. Soc. B, 289-300.
2. Benjamini, Y., Liu, W. (1999). A step-down multiple hypotheses testing procedure that controls the false discovery rate under independence. JSPI, 163-170.
3. Benjamini, Y., Liu, W. (2001). A distribution-free multiple-test procedure that controls the false discovery rate. Manuscript available from FDR Website of Y. Benjamini.
4. Benjamini, Y., Yekutieli, D. (2001). The control of the false discovery rate in multiple testing under dependency. Annals of Statistics **29**, 1165-1188.
5. Benjamini, Y., Krieger, A., and Yekutieli, D. (2001). Two stage linear step up FDR controlling procedure. Manuscript available from FDR Website of Y. Benjamini.
6. Dunnett, C. W. (1955). A multiple comparison procedure for comparing several treatments with a control. J. Amer. Statist. Assoc. , **60**, 1096-1121.
7. Dunnett, C. W.,and Tamhane, A. C. (1991). Step-down multiple tests for comparing treatments with a control in unbalanced one-way layouts. Statistics in Medicine, **10**, 939-947.
8. Dunnett, C. W.,and Tamhane, A. C. (1992). A step-up multiple test procedure. J. Amer. Statist. Assoc. **87**, 162-170.
9. Horn, M., Dunnett, C. W. (2004). Power and sample size comparisons of stepwise FWE and FDR controlling test procedures in the normal many-one case. Recent Developments in

Multiple Comparison, edited by Y. Benjamini, S. Sarkar and F. Bretz. IMS Lecture Notes Monograph Series. 48 - 64.

10. Korn, Edward L., Troendle, James F., McShane, Lisa M. and Simon, Richard (2003). Controlling the number of false discoveries: Application to high-dimensional genomic data. Journal of Statistical Planning and Inference **124** issue 2,379 - 398.
11. Naik, U. D. (1975). Some selection rules for comparing p processes with a standard. Commun. Statist., Ser. A , **4** , 519-535.
12. Sarkar, S.K. (2002). Some results on false discovery rate in stepwise multiple testing, Annals of Statistics **30**, 239-257.
13. Scheffe H. (1953). A method for judging all contrasts in the analysis of variance. Biometrika **40**, 87-104.
14. Somerville, Paul N. (2003). "Optimum" FDR procedures and MCV values. Technical report TR-03-01 Department of Statistics, University of Central Florida.
15. Somerville, Paul N. (2004a) A step-down procedures for large numbers of hypotheses. Paper presented at the annual meetings of the Florida Chapter of the ASA.
16. Somerville, Paul N. (2004b). FDR step-down and step-up procedures for the correlated case. Recent Developments in Multiple Comparison, edited by Y. Benjamini, S. Sarkar and F. Bretz. IMS Lecture Notes Monograph Series. 100 -118.
17. Tukey, J. W. (1953). The problem of multiple comparisons. Department of Statistics, Princeton University.
18. van der Laan, Mark J., Dudoit, Sandrine and Pollard, Katherine S. (2004). Multiple testing. Part II. Step-down procedures for control of the generalized family-wise error rate. Statistical Applications in Genetics and Molecular Biology: Vol. 3: No. 1, Article 13.

Applying Security Engineering to Build Security Countermeasures: An Introduction

Organizers: Tai-hoonn Kim[1] and Ho-yeol Kwon[2]

[1] San-7, Geoyeo-Dong, Songpa-Gu, Seoul, Korea
taihoon@empal.com
[2] Gangwon University, 192-1, Hyoja2-Dong
Chunchon, Kangwon-Do, 200-701, Korea
kwh@object.cau.ac.kr

Abstract. The general systems of today are composed of a number of components such as servers and clients, protocols, services, and so on. Systems connected to network have become more complex and wide, but the researches for the systems are focused on the 'performance' or 'efficiency'. While most of the attention in system security has been focused on encryption technology and protocols for securing the data transaction, it is critical to note that a weakness (or security hole) in any one of the components may comprise whole system. Security engineering is needed for reducing security holes may be included in the software. Therefore, more security-related researches are needed to reduce security weakness may be included in the software. This paper introduces some methods for reducing the threat to the system by applying security engineering, and proposes a method for building security countermeasure.

1 Introduction

Many products, systems, and services are needed and used to protect information. The focus of security engineering has expanded from one primarily concerned with safeguarding classified government data to broader applications including financial transactions, contractual agreements, personal information, and the Internet. These trends have elevated the importance of security engineering [1]. There are many security related standards, methods and approaches, and their common objective is to assure the security of IT system or software products. ISO/IEC TR 15504, the Software Process Improvement Capability Determination (SPICE), provides a framework for the assessment of software processes, and it has some considerations for security although they are relatively poor to others. For example, the considerations for security related to software development and developer are lacked. Considerations for security are expressed well in some evaluation criteria such as ISO/IEC 21827, the Systems Security Engineering Capability Maturity Model (SSE-CMM), and ISO/IEC 15408, Common Criteria (CC) [2][3][4][5][6]. It is essential that not only the customer's requirements for software functionality should be satisfied but also the security requirements imposed on the software development should be effectively analyzed and implemented in contributing to the security objectives of customer's requirements. Unless suitable requirements are established at the start of the software development process, the resulting end product, however well engineered, may not meet the objectives of its anticipated consumers. The IT products like as firewall, IDS (Intrusion Detection System) and VPN (Virtual Private Network) are made to perform special functions related to security, and used to supply security characteristics. But the method using these products may be not the perfect solution. Therefore, when making some kinds of software products, security-related requirements must be considered.

2 A Summary of Security Engineering

Security engineering is focused on the security requirements for implementing security in software or related systems. In fact, the scope of security engineering is very wide and encompasses:

- the security engineering activities for a secure software or a trusted system addressing the complete lifecycle of: concept definition, analysis of customer's requirements, high level design and low level design, development, integration, installation and generation, operation, maintenance end de-commissioning,
- requirements for product developers, secure systems developers and integrators, organizations that develop software and provide computer security services and computer security engineering,
- applies to all types and sizes of security engineering organizations from commercial to government and the academe.

The security engineering should not be practiced in isolation from other engineering disciplines. Maybe the security engineering promotes such integration, taking the view that security is pervasive across all engineering disciplines (e.g., systems, software and hardware) and defining components of the model to address such concerns. The main interest of customers and suppliers may be not improvement of the development of security characteristics but performance and functionality. If developers consider some security-related aspects of software developed, maybe the price or fee of software more expensive. But if they think about that a security hole can compromise whole system, some cost-up will be appropriate.

3 Application of Security Engineering

A wide variety of organizations can apply security engineering to their work such as the development of computer programs, software and middleware of applications programs or the security policy of organizations. Therefore, appropriate approaches, methods and practices are needed for product developers, service providers, system integrators and system administrators even though they are not security specialists. Some of these organizations deal with high-level issues (e.g., ones dealing with operational use or system architecture), others focus on low-level issues (e.g., mechanism selection or design), and some do both. The security engineering may be applied to all kinds of organizations. Use of the security engineering principle should not imply that one focus is better than another is or that any of these uses are required. An organization's business focus need not be biased by use of the security engineering. Based on the focus of the organization, some, but not all, of approaches or methods of security engineering may be applied very well. In fact, generally, it is true that some of approaches or methods of security engineering can be applied to increase assurance level of software. Next examples illustrate ways in which the security engineering may be applied to software, systems, facilities development and operation by a variety of different organizations. Security service provider may use security engineering to measure the capability of an organization that performs risk assessments. During software or system development or integration, one would need to assess the organization with regard to its ability to determine and analyze security vulnerabilities and assess the operational impacts. In the operational case, one would need to assess the organization with regard to its ability to monitor the security posture of the system, identify and analyze security vulnerabilities, and assess the operational impacts. Countermeasure Developers may use security engineering to address determining and analyzing security vulnerabilities, assessing operational impacts, and providing input and guidance to other groups involved (such as a software group). Software or product developers may use security engineering to gain an understanding of the customer's security needs and append security requirements to the customer's requirements. Interaction with the customer is required to ascertain them. In the case of a product, the customer is generic as the product is developed a priori independent of a specific customer. When this is the case, the product-marketing group or another group can be used as the hypothetical customer, if one is required. The main objective of application of security engineering is to provide assurance about the software or system to customer, and the assurance level of a software or system may be the critical factor has influence on deciding purchase. Therefore, the meaning of the application of security engineering to the software is the application of some assurance methods to the software development lifecycle phases. Assurance methods are classified in Fig.1 [7]. Depending on the type of assurance method, the assurance gained is based on the aspect assessed and the lifecycle phase. Assurance approaches yield different assurance due to the deliverable (IT component or service) aspect examined. Some approaches examine different phases of the deliverable lifecycle while others examine the processes that produce the deliverable (indirect examination of the deliverable). Assurance approaches include facility, development, analysis, testing, flaw remediation, operational, warranties, personnel, etc. These assurance approaches can be further broken down; for example, testing assurance approach includes general testing and strict conformance testing assurance methods.

4 Applying Security Engineering to Build Countermeasures

In general, threat agents' primary goals may fall into three categories: unauthorized access, unauthorized modification or destruction of important information, and denial of authorized access. Security countermeasures are implemented to prevent threat agents from successfully achieving these goals. Security countermeasures should be considered with consideration of applicable threats and security solutions deployed to support appropriate security services and objectives. Subsequently, proposed security solutions may be evaluated to determine if residual vulnerabilities exist, and a managed approach to mitigating risks may be proposed. Countermeasures must be considered and designed

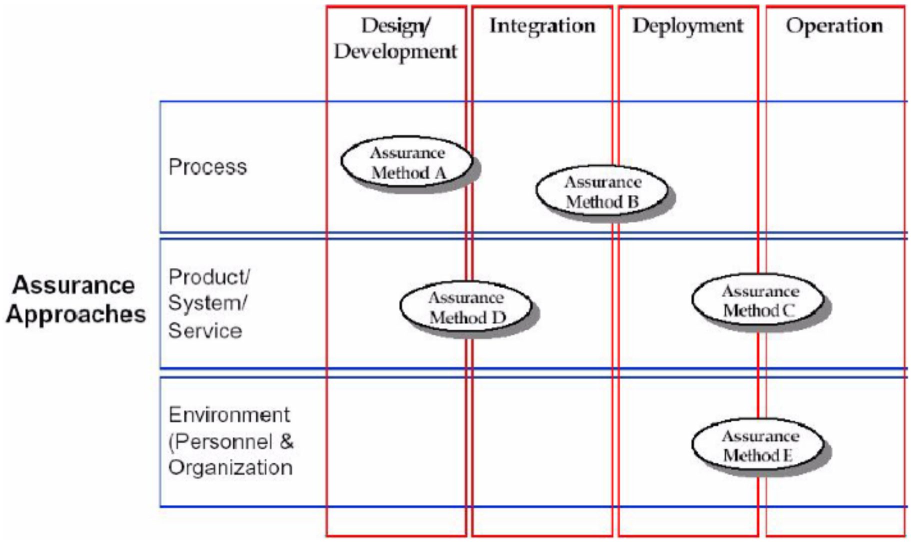


Fig. 1. Categorization of existing assurance methods

from the starting point of some IT system design or software development processes. The countermeasure or a group of countermeasures selected by designers or administrators may cover all the possibility of threats. But a problem exits in this situation. How and who can guarantee that the countermeasure is believable? Security engineering may be used to solve this problem. In fact, the processes for building of security countermeasures may not fixed because the circumstances of each IT system may be different. We propose a method for building security countermeasures as below.

4.1 Threats Identification

A ’threat’ is an undesirable event, which may be characterized in terms of a threat agent (or attacker), a presumed attack method, a motivation of attack, an identification of the information or systems under attack, and so on. Threat agents come from various backgrounds and have a wide range of financial resources at their disposal. Typically Threat agents are thought of as having malicious intent. However, in the context of system and information security and protection, it is also important to consider the threat posed by those without malicious intent. Threat agents may be Nation States, Hackers, Terrorists or Cyber terrorists, Organized Crime, Other Criminal Elements, International Press, Industrial Competitors, Disgruntled Employees, and so on. Most attacks maybe aim at getting inside of information system, and individual motivations of attacks to "get inside" are many and varied. Persons who have malicious intent and wish to achieve commercial, military, or personal gain are known as hackers (or cracker). At the opposite end of the spectrum are persons who compromise the network accidentally. Hackers range from the inexperienced Script Kiddy to the highly technical expert.

4.2 Determination of Robustness Strategy

The robustness strategy is intended for application in the development of a security solution. An integral part of the process is determining the recommended strength and degree of assurance for proposed security services and mechanisms that become part of the solution set. The strength and assurance features provide the basis for the selection of the proposed mechanisms and a means of evaluating the products that implement those mechanisms. Robustness strategy should be applied to all components of a solution, both products and systems, to determine the robustness of configured systems and their component parts. It applies to commercial off-the-shelf (COTS), government off-the-shelf (GOTS), and hybrid solutions. The process is to be used by security requirements developers, decision makers, information systems security engineers, customers, and others involved in the solution life cycle. Clearly, if a solution component is modified, or threat levels or the value of information changes, risk must be reassessed with respect to the new configuration. Various risk factors, such as the degree of damage that would be suffered if the security policy were violated, threat environment, and so on, will be used to guide determination of an appropriate strength and an associated level of assurance for each mechanism. Specifically, the value of the information to be protected and the perceived threat environment are used to obtain guidance on the recommended strength of mechanism level (SML) and evaluation assurance level (EAL).

4.3 Consideration of Strength of Mechanisms

SML (Strength of Mechanism Levels) are focusing on specific security services. There are a number of security mechanisms that may be appropriate for providing some security services. To provide adequate information security countermeasures, selection of the desired (or sufficient) mechanisms by considering particular situation is needed. An effective security solution will result only from the proper application of security engineering skills to specific operational and threat situations. The strategy does offer a methodology for structuring a more detailed analysis. The security services itemized in these tables have several supporting services that may result in recommendations for inclusion of additional security mechanisms and techniques.

4.4 Selection of Security Services

In general, primary security services are divided five areas: access control, confidentiality, integrity, availability, and non-repudiation. But in practice, none of these security services is isolated from or independent of the other services. Each service interacts with and depends on the others. For example, access control is of limited value unless preceded by some type of authorization process. One cannot protect information and information systems from unauthorized entities if one cannot determine whether that entity one is communicating with is authorized. In actual implementations, lines between the security services also are blurred by the use of mechanisms that support more than one service.

4.5 Application of Security Technologies

An overview of technical security countermeasures would not be complete without at least a high-level description of the widely used technologies underlying those countermeasures. Next items are some examples of security technologies.

- Application Layer Guard, Application Program Interface (API),
- Common Data Security Architecture (CDSA),
- Internet Protocol Security (IPSec), Internet Key Exchange (IKE) Protocol,
- Hardware Tokens, PKI, SSL, S/MIME, SOCKS,
- Intrusion and Penetration Detection.

4.6 Determination of Assurance Level

The discussion of the need to view strength of mechanisms from an overall system security solution perspective is also relevant to level of assurance. While an underlying methodology is offered by a number of ways, a real solution (or security product) can only be deemed effective after a detailed review and analysis that consider the specific operational conditions and threat situations and the system context for the solution. Assurance is the measure of confidence in the ability of the security features and architecture of an automated information system to appropriately mediate access and enforce the security policy. Evaluation is the traditional method ensures the confidence. Therefore, there are many evaluation methods and criteria exist. In these days, many evaluation criteria such as ITSEC are replaced by the Common Criteria. The Common Criteria provide assurance through active investigation. Such investigation is an evaluation of the actual product or system to determine its actual security properties. The Common Criteria philosophy assumes that greater assurance results come from greater evaluation efforts in terms of scope, depth, and rigor.

5 Conclusions and Future Work

As mentioned earlier, security should be considered at the starting point of all the development processes. Making an additional remark, security should be implemented and applied to the IT system or software products by using the security engineering. This paper introduces some methods for reducing the threat to the system by applying security engineering, and proposes a method for building security countermeasure. But this method we proposed can't cover all the cases. Therefore, more detailed research is needed. And the research for generalizing these processes may be proceeded, too.

References

1. ISO. ISO/IEC 21827 Information technology - Systems Security Engineering Capability Maturity Model (SSE-CMM)
2. ISO. ISO/IEC TR 15504-2:1998 Information technology - Software process assessment - Part 2: A reference model for processes and process capability
3. ISO. ISO/IEC TR 15504-5:1998 Information technology - Software process assessment - Part 5: An assessment model and indicator guidance
4. ISO. ISO/IEC 15408-1:1999 Information technology - Security techniques - Evaluation criteria for IT security - Part 1: Introduction and general model
5. ISO. ISO/IEC 15408-2:1999 Information technology - Security techniques - Evaluation criteria for IT security - Part 2: Security functional requirements

6. ISO. ISO/IEC 15408-3:1999 Information technology - Security techniques - Evaluation criteria for IT security - Part 3: Security assurance requirements
7. Tai-Hoon, Kim: Approaches and Methods of Security Engineering, ICCMSE 2004
8. Tai-Hoon Kim, Byung-Gyu No, Dong-chun Lee: Threat Description for the PP by Using the Concept of the Assets Protected by TOE, ICCS 2003, LNCS 2660, Part 4, pp. 605-613
9. Tai-hoon Kim, Tae-seung Lee, Kyu-min Cho, Koung-goo Lee: The Comparison Between The Level of Process Model and The Evaluation Assurance Level. The Journal of The Information Assurance, Vol.2, No.2, KIAS (2002)
10. Tai-hoon Kim, Yune-gie Sung, Kyu-min Cho, Sang-ho Kim, Byung-gyu No: A Study on The Efficiency Elevation Method of IT Security System Evaluation via Process Improvement, The Journal of The Information Assurance, Vol.3, No.1, KIAS (2003)
11. Tai-hoon Kim, Tae-seung Lee, Min-chul Kim, Sun-mi Kim: Relationship Between Assurance Class of CC and Product Development Process, The 6th Conference on Software Engineering Technology, SETC (2003)
12. Ho-Jun Shin, Haeng-Kon Kim, Tai-Hoon Kim, Sang-Ho Kim: A study on the Requirement Analysis for Lifecycle based on Common Criteria, Proceedings of The 30th KISS Spring Conference, KISS (2003)
13. Tai-Hoon Kim, Byung-Gyu No, Dong-chun Lee: Threat Description for the PP by Using the Concept of the Assets Protected by TOE, ICCS 2003, LNCS 2660, Part 4, pp. 605-613
14. Haeng-Kon Kim, Tai-Hoon Kim, Jae-sung Kim: Reliability Assurance in Development Process for TOE on the Common Criteria, 1st ACIS International Conference on SERA

CC-SEMS: A CC Based Information System Security Evaluation Management System

Young-whan Bang, Yeun-hee Kang, and Gang-soo Lee

Hannam University, Dept. of Computer Science
Daejon, 306-791, Korea
{bangyh@se,dusi82@se,gslee@eve}.hannam.ac.kr

Abstract. Project of Common Criteria (CC) based evaluations of an IT security product/system takes so much cost and time that effective evaluation project management is strongly required. We present a CC based security evaluation management system (CC-SEMS) that is useful for evaluation facilities and developers of IT security product. Evaluation activity program, evaluation plan, XML-DTD of deliverable, management object, and evaluation DB are proposed as distinct approaches of CC-SEMS.

1 Introduction

Various types of information security products have been developed, evaluated and certified since middle of 1980's under various evaluation criteria/manual such as TCSEC [1], ITSEC/ITSEM [2,3], CTCPEC [4] and CC/CEM [5,6]. CC (Common Criteria), pronounced as ISO/IEC 15408, is an integrated evaluation criteria of the others.

Most evaluations of product (i.e., Target of Evaluation: TOE), are completed within an optimum evaluation period and few hundreds of thousand dollars from starting. Others, of a more complex nature, can take much longer and more expensive depending on the target Evaluation Assurance Level (EAL) and Security Target (ST) that is a security requirement specification. Additionally, the elapsed time and cost for the security evaluation process is not only dependence on the availability of the correct developer documentation(or deliverable), but also the nature and complexity of the software, and whether any re-use can be made of work performed from previous evaluations of the same TOE [7].

An evaluation manual such as ITSEM and CEM is overall guidance of evaluation [3,6,7]. An evaluator in an evaluation facility should refer the manuals. ITSEM is a high-level evaluation manual for ITSEC. ITSEM have only one Evaluation Activity Program (EAP). We need 7 kinds of EAP per each EAL. CEM has two problems. First, it is so abstract and high-level evaluation guide that it is un-useful for evaluators. In fact, CEM is merely a re-editing document of CC by context of "evaluator actions" statements in CC. Second, it is not providing guidance for EAL 5 through 7, nor for evaluations using other assurance packages. Thus we need a practical evaluation guidance, automation of evaluation activity, and a management system of evaluation projects.

Guidance for accreditation and operation of evaluation facilities or test laboratories has been internationally or domestically standardized such as EN45001, ISO/IEC 17025, NIST Handbook 150, 150-20, CCEVS SP # 1, SP # 4, Canada's PALCAN, United Kingdom's UK-SP2. Those cover, however, merely accreditation and operation of evaluation facilities. A detailed and practical evaluation guidance such as evaluation project program and management, that is useful to evaluators in evaluation facilities, is strongly required.

An evaluation facility should operate an efficient CC Security Evaluation Management System (CC-SEMS) on their evaluation environment for the purpose of cooperative and concurrent managing the evaluation resource (e.g., evaluator, tool, deliverable, and criteria). CC-SEMS is integrated application principles of project management [8], workflow management [9], process management [10] and web service technologies [11].

Fig.1 presents architecture of CC-SEMS. Note that EAP template, Evaluation Plan and XML-DTD of deliverables, those are included in the *preparation time*, should be prepared before a start of evaluation. The right part of Fig.1 (i.e., management of Management Object and control engine) is operated in the *evaluation time*. In the next section, we present EAP template, Evaluation Plan and XML/DTD of deliverables in *preparation time* of CC-SEMS. Section 3 presents management object, Evaluation DB and control engine in *evaluation time*. Finally, we implement, analyze and conclude in Section 4.

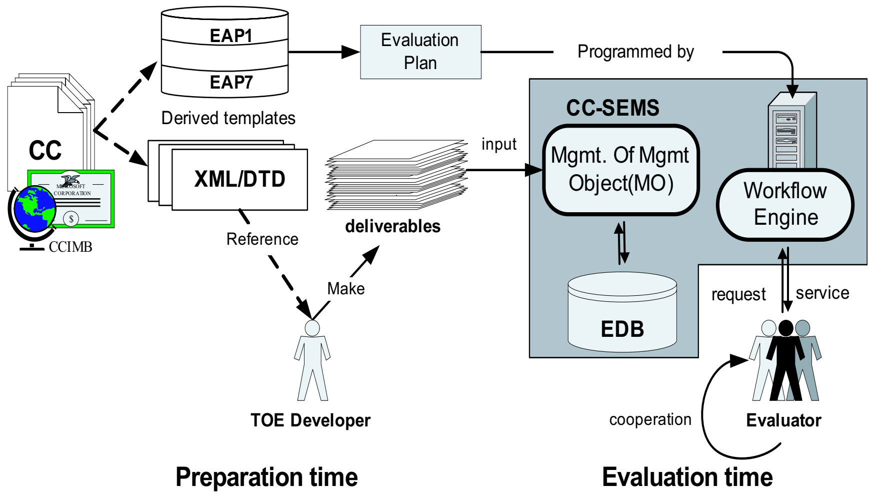


Fig. 1. Architecture of CC-SEMS

2 Preparation Time of CC-SEMS

A preparation time refers an activity of preparation of evaluation. Evaluation Plan and deliverable should be written by an evaluation project manager and developer, respectively.

2.1 Evaluation Activity Programs Templates and Evaluation Plan

Evaluation Activity Programs(EAP) Templates: Dependency list is defined for each functional and assurance components in CC. Dependencies among "functional components" arise when a component is not self-sufficient and relies upon the functionality of, or interaction with, another component for its own proper functioning.

Additionally, dependencies among assurance components arise when a component is not self-sufficient, and relies upon the presence of another component. Recall that assurance class, component, evaluator action and "developer action and content and presentation of evidence" correspond to activity, sub-activity, action and "work unit" in an evaluation project, respectively [6].

We regard dependence relations between assurance components as precedence relationship between evaluation sub-activities in context of evaluation project. We drive EAP templates, i.e., EAP1 through EAP7, which correspond to EAL1 through EAL7, from dependency relationship among assurance components. Note that the word "template" means what is independent to a TOE and an evaluation environment. Templates are only derived from CC. EAP is defined as follows:

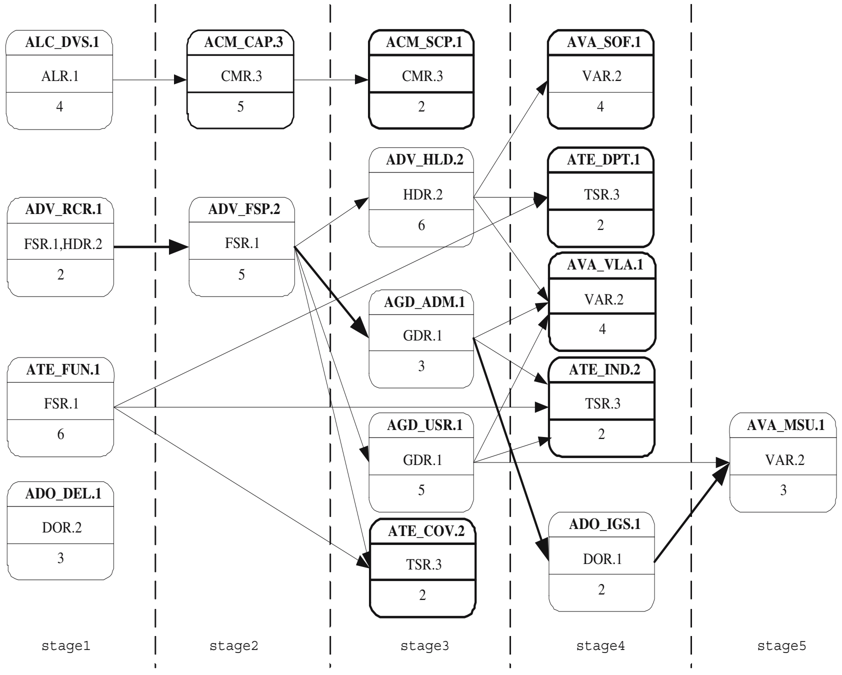
```
S# = 1;    // stage number
Temp = Original; // Temp is a copy of Original (i.e., list of dependency relation defined in EALi).
Foreach component COM in Temp that don't have 'in-going' dependency relation,
  do
     Draw the COM as an AN on stage P of EAP;    //  AN is an activity node.
     Delete COM's "out-going" dependent relations from Temp;
  end-do
repeat
  for each component COM in Temp that don't have 'in-going' dependent relation,
     do
      Draw the COM as an AN in stage Q of EAP;
          Delete the COM's "out-going" dependent relations from Temp;
          Draw arcs from AN in P to AN in Q by using Original;
     end-do
P = Q; S# = S# + 1;
until(there is no more relation in Original.)
if(A has an arc to B) and (B has an arc to C) and (A has an arc to C),
  then delete arc that is from A to C.   // A, B, C are AN in EAP. If there is 'transitive relationship'
  in EAP, then delete them for the purpose of simplification.
```

Fig. 2. Evaluation activity program generation algorithm

EAP = (AN, ARC, STG)

- AN is a set of *activity nodes*. AN has three attributes such as *component name, deliverable name* and *assignment*. (e.g., duration, cost, developer, tool). An AN presents an evaluation activity and necessary deliverables of an assurance component (see Table 1), as well as assignment of resource such as duration, cost, developer and tool. Note that assignment is empty in *template* of EAP. Assignment attribute, especially, is used for the purpose of evaluation project management.
- ARC is a set of *arcs* from an AN to another AN in other stage. An ARC presents precedence relationships between two ANs. ANs in a stage can be concurrently processed alone with the project management.
- STG is a set of *stages*. A pass from stage1 to final stage is total ordering of AN.

- EAP is produced by means of an "EAP generation algorithm" as shows in Fig.2. The algorithm generates a total ordering from partial orderings. An EAP template is presented by an activity network or Gantt-chart. Fig.3 shows an activity network presentation of an EAP3 template.



(Note: Numerical values are basic values that will be changed according to evaluation environment)

Fig. 3. An activity network presentation of EAP3 template and an EP

Evaluation Plan (EP): An EP is realization, customization or instantiation of one of EAP templates. An evaluation project manager shell plan his evaluation project by using EAP template and scheduling algorithm of resources such as time, stakeholder, tool and cost.

Fig.3 present an EP for EAL3 of some TOE. Note that assignment attribute values are filled in template. In Fig.3, the value can be interpreted as evaluation duration (weeks or months), cost ($), evaluator identifier, tool id., and so on. Numerical values(i.e. weeks/month, cost) in EAP are basic evaluation duration(week, month) or cost, that derived from CC and CEM. If evaluation environment is changed, the values in the EAP are also changed by the same ratio. An EP presented by a Gantt chart as shows in Fig.4. An EP can be scheduled by the following policies:

- *Multi-stage waterfall type scheduling* : It is simple and un-optimal. An EP of Fig.3 takes 24 weeks by means of this policy.

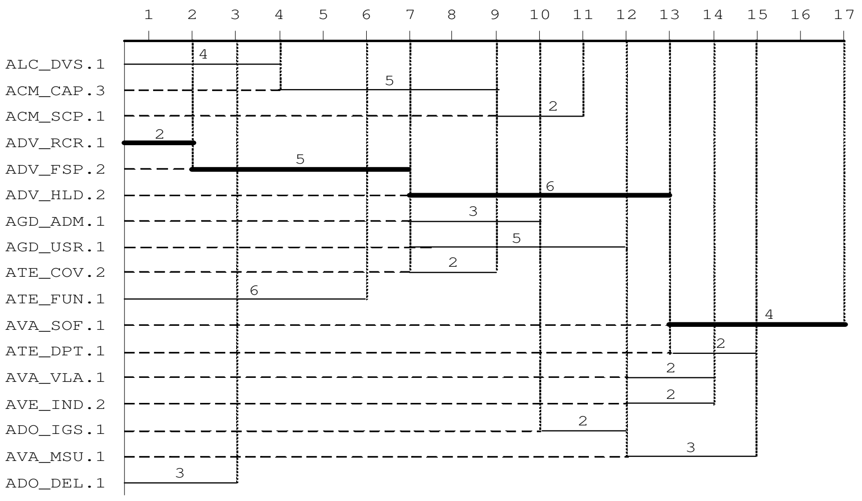


Fig. 4. Optimal scheduling of EP of EAP3

- *Optimal scheduling* : In Fig.4, the critical evaluation activity path is ADV_RC R.1→ ADV_FSP.2 → ADV_HLD.2 → AVA_SOF.1. Total evaluation duration is 17 weeks. If we want to shortening the evaluation duration, we must be shorten the duration of the evaluation activities on the critical path.

2.2 Deliverables

Necessary content of deliverables: Deliverables are produced by developer of TOE and consumed by evaluator. Deliverables that are presented by a document or source code is image of development process and a product. It is important to note that a TOE includes not only development process, but a product itself. We derive a list and a content of deliverables for each AN in EAP1 through EAP7 templates by using "deliverable generation algorithm". The algorithm has two phases - derivation and refinement of necessary contents.

1. *Derivation of necessary contents*: Necessary contents in deliverable (or document) for evaluation of assurance component can be derived from "developer action elements" and "content and presentation of evidence elements" in assurance components. Each class, family and component in CC assurance requirement is mapped to title, chapter and section of a deliverables, respectively.
2. *Refinement of necessary contents*: If a document is derived from an assurance class, and then 13 documents are needed. Thus we need 91(13×7) types, in maximum, of documents. Those are redundant and too short or too long document. Thus we should not only minimize the amount of deliverables (or document), but also assure the consistency among documents. The more redundant documents are, the less consistent among documents might be.

We can reduce volumes of documents by using the following rules:

- *CC conformance rule*: A specific document includes a minimal content and presentation of evidence elements that specified in CC
- *Subset rule*: A document for upper EAL includes those of lower EAL's.
- *Merge Rule*: Two or more documents for assurance classes that are small or redundant, may be merged into one document.
- *Minimal Rule*: Two or more documents for assurance classes, which contents are the same, may be merged into one document.

Table 1. List of deliverables

Deliverables(documents)	EAL1	EAL2	EAL3	EAL4	EAL5	EAL6	EAL7	Corresponding assurance class-family
Security target report (STR)	STR.1							ASE
Configuration management Report (CMR)	CMR.1	CMR.2	CMR.3	CMR.4	CMR.5	CMR.6		ACM
Delivery and operation report (DOR)	DOR.1	DOR.2		DOR.3			DOR.4	ADO
Functional Specification (FSR)	FSR.1			FSR.2	FSR.3		FSR.4	ADV_FSP, ADV_RCR
High-level design specification (HDR)		HDR.1	HDR.2		HDR.3	HDR.4	HDR.5	ADV_HLD, ADV_RCR
Implementation report (IMR)				IMR.1	IMR.2	IMR.3	IMR.4	ADV_IMP, ADV_RCR
Structural specification (INR)					INR.1	INR.2	INR.3	ADV_INT, ADV_RCR
Low-level design specification (LDR)				LDR.1	LDR.2	LDR.3	LDR.4	ADV_LLD, ADV_RCR
Security policy sped (SPR)				SPR.1	SPR.2		SPR.3	ADV_SPM
Guidance document (GDR)	GDR.1							AGD
Lifecycle support report (ALR)			ALR.1	ALR.2	ALR.3	ALR.4	ALR.5	ALC
Test report (TSR)	TSR.1	TSR.2	TSR.3		TSR.4	TSR.5	TSR.6	ATE
Vulnerability analysis report (VAR)		VAR.1	VAR.2	VAR.3	VAR.4	VAR.5		AVA

We can reduce numbers of deliverables to 51 (from 91) by using the rules as shown in Table 1. Our approach is more useful and practical than FAA's Data Item Description[13].

XML-DTD for Deliverables: Content and structure (or schema) of deliverables are defined by XML-DTD. XML-DTD is regarded as syntax of deliverable. Each section or paragraph of a deliverable is presented by tag in XML. XML-DTD has many advantages in web based evaluation environment as follows:

- Standardization of structure of deliverable
- Automatic syntax and consistency checking by using XML processors
- Automatic handling of deliverables by XML based application

A producer (i.e., developer or sponsor) of deliverable, firstly, downloads a XML-DTD file of a deliverable from an evaluation web server of CC-SEMS. Then, he makes a XML file by using the loaded file and his own content; finally, he submits the XML file to an *evaluation web server* of CC-SEMS. In the server, a XML processor, firstly, will checks syntax and consistency of the submitted XML formed deliverables. Then, a

```
<!ELEMENT CMR (CMR_1_eva_num, CMR_2_intro, CMR_3_conf_item, CMR_R_CMS, CMR_5_CMP, CMR_6_reference)>
<!ELEMENT CMR_1_eva_num (CMR_1_1_identity)>
    <!ELEMENT CMR_1_1_identity (#PCDATA)>
 <!ELEMENT CMR_2_intro (CMR_2_1_identity, CMR_2_2_concept)>
    <!ELEMENT CMR_2_1_identity (CMR_2_1_1_ProductID, CMR_2_1_2_ProductName, CMR_2_1_3_DocID,
                               CMR_2_1_4_DocName,  CMR_2_1_5_PubNo)>
        <!ELEMENT CMR_2_1_1_ProductID (#PCDATA)>
        <!ELEMENT CMR_2_1_2_ProductName (#PCDATA)>
        <!ELEMENT CMR_2_1_3_DocID (#PCDATA)>
        <!ELEMENT CMR_2_1_4_DocName (#PCDATA)>
        <!ELEMENT CMR_2_1_5_PubNo (#PCDATA)>
                                                    --- omitted below ----
```

Fig. 5. XML-DTD of Configuration Management Report (CMR.4) for EAL4

part (i.e., chapter tag or section tag) of deliverable is allocated to a scheduled evaluation activity by CC-SEMS. We define XML-DTDs for deliverables listed in Table 1. Fig.5 presents one of XML-DTD.

3 Evaluation Time of CC-SEMS

3.1 Management Object (MO)

MO, that is resource or target of management, is defined as follows:

MO = (STKE, DELI, TOOL, TIME, COST, CRIT), where,

- STKE is stakeholder or interested party of CC-SEMS. TOE produce, evaluation participant and evaluation users are subtypes of STKE [12]: *TOE producer* is a *sponsor, developer* or *consultant*. *Evaluation participant* is a *evaluator, certifier* and *validator*. Note that certifier and validator are working in a certification/validation body. *CC evaluator* an *evaluator, approver* or *accredit* or who is working in an evaluation facilities. Finally, *evaluation user* is a consumer or *product vendor*.
- DELI is deliverables or documents. Subtypes of DELI are input (i.e., an image of TOE), process (temporal doc.) and output (results doc.).
- TOOL is a given evaluation *tool* such as tester program.
- TIME is a given evaluation *time* duration (e.g., 10 months).
- COST is a given evaluation *cost* (e.g., 1 million dollars).
- CRIT is evaluation *criteria* such as CC or PP(Protection Profile) or ST. Note that PP or ST can be regarded as evaluation criteria.

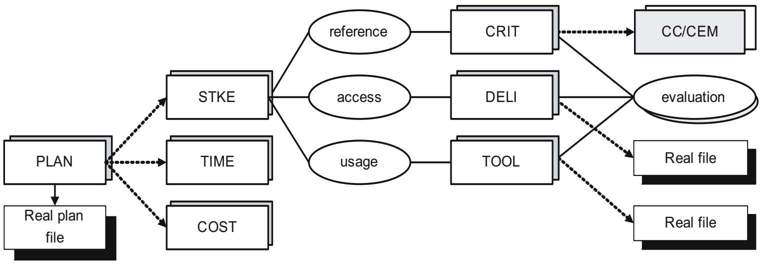


Fig. 6. A Schema of EDB

3.2 WorkFlow Engine (WFE)

MO is controlled by the WFE or manager by way of a project plan. Inputs of WFE are EDB, an EP, and MO (e.g., evaluators, deliverables). WFE allocates or deallocates deliverables to evaluator, controls access to deliverables, conforms some work to evaluator, and so on. Outputs of WFE are control signal/message to evaluator or developer by means of online or offline.

3.3 Evaluation DB (EDB)

EDB is used for storing state of MO of an evaluation project. EDB Schema, that is a logical structure of EDB, is entities of MO and relations among the entities as shows in Fig.6. A *state* of MO means a state of evaluation project, in other word, current value (snap shot) of an EDB that has MO's attribute value. A state is transited to other state when an event (e.g., 'testing is finished') is occurred on evaluation time.

Thus, processing activities of a state of MO are: identify, measurement and monitoring the MO's attributes values; store and update EDB by means of manual or automatic. Tables, a data structure for entity and relation in EDB, are organized as below (Italic word is primary key, underline word is foreign key).

- *Entity tables and attributes*: PLAN(*plan_id*, proj_id, proj_information, link to real plan file), STKE(*stke_id*, type, name, address, major, cost, etc.), DELI(*deli_id*, type, name, content, link to real file, etc.), TOOL(*tool_id*, type, cost, function, location, link to real file, etc.), COST(*account_id, amount*, allocated, etc.), TIME(*time_id*, etc.), CRIT(link to real CC/CEM, etc.).
- *Relation tables and attributes*: access(stke_id, deli_id, *access_id*, access_type, access_log, etc.), access_type = (creation, update, read, approve, etc.),usage (stke_id, tool_id, *usage_id*, usage_type, usage_log, etc.), usage_type = (using, update), reference(stke_id, crit_id, *refe_id*, ref_log, etc.), evaluation (crit_id, del_id, *eval_id*, tool_id, etc.).

4 Implementation, Analysis and Conclusion

Prototype version of CC-SEMS has been implemented by using Windows, JBuilder 9.0, MySQL 3.23.38 and PHP v.0.6. Currently we are developing upper version. CC-SEMS is

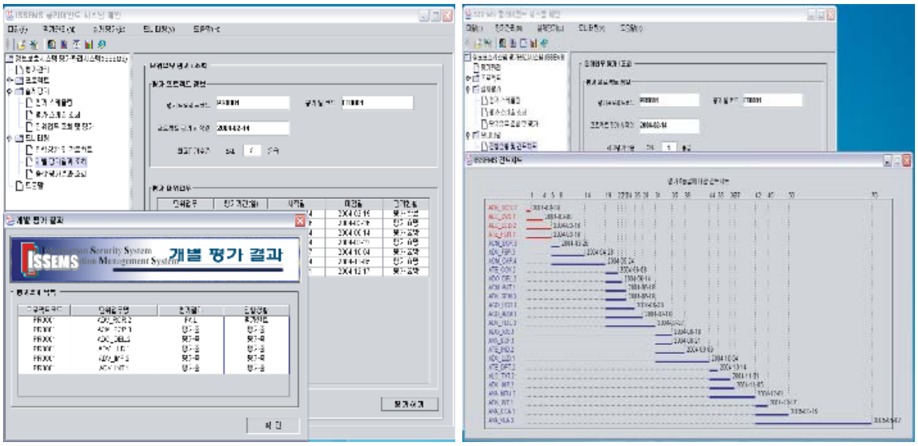


Fig. 7. Some screen shots of CC-SEMS (Korean version)

consisted of EDB server, application server, client and web server. Note that XML-DTD files of deliverables are uploaded on the web server.

We have developed a Security Target(ST) of CC-SEMS because CC-SEMS is also an IT security product. Recall that ST is a requirement specification in context of security function and assurance [14,15]. Fig.7 presents Korean version of CC-SEMS.

CC-SEMS is not only an integrated approach from project management, groupware, workflow management, documentation engineering, XML and web technologies, but also has following distinct and useful approaches:

- *Necessary content of deliverables*, that derived from "developer action elements" and "content and presentation of evidence elements" in CC, and some *rules for reducing volumes* of deliverables are expected to be useful at CC evaluation schemes.
- *XML-DTD* of all sort of deliverables are useful for applying Web services in security evaluation.
- *Seven EWP templates*, derived from dependency relations in components of CC, are useful for developing an Evaluation Plan for evaluation project scheduling in evaluation facilities.
- *Management object* and *evaluation DB schema* are useful for modeling and developing any other evaluation management systems.

The authors can not find yet any other similar approach or system to CC-SEMS in CC evaluation community. The system is ongoing development project. Thus we will formalize our approaches and upgrade the system by using actual data and problems that are occurred in real application of the system.

Acknowledgement

This work was supported by a grant No. R12-2003-004-01001-0 from KOSEF.

References

1. DoD. Trusted Computer System Evaluation Criteria (TCSEC), Dec. 1985.
2. European Communication, "Information Security Evaluation Criteria (ITSEC)", Ver. 1.2, June 1991.
3. European Community, Information Technology Security Evaluation Criteria (ITSEM), Ver 1.0, 1993.
4. Canadian System Security Centre, , "The Canadian Trusted Computer Product Evaluation Criteria (CTCPEC)", Ver.3e, Jan. 1993.
5. CC, Common Criteria for Information Technology Security Evaluation, CCIMB-2004-03, Version 2.2, Jan. 2004.
6. Common Methodology for Information Technology Security Evaluation (CEM), CCIMB-2004-01-04, Version 2.2, Jan. 2004.
7. Common Criteria for Information Technology Security Evaluation, User Guide, Oct. 1999.
8. W. Royce, Software project management - A Unified Framework, AW, 1998.
9. P. Lawrence (ed.), Workflow handbook - 1997, John Wiley & Sons, 1997.
10. A. Fuggetta and A. Wolf (ed.). Software Process, John Wiley & Sons, 1996.
11. G. Alonso, et al., Web services - Concept, Architectures and Applications, Springer, 2004.
12. R. Priet-Diaz, The CC Evaluation Process - Process Explanation, Shortcomings, and Research Opportunities, Commonwealth Information Security Center Technical Report CISC-TR-2002-003, James Madison University, Dec. 2002.
13. Data Item Description, Federal Aviation Administration, www.faa.gov/aio/cuiefsci/PP-library/index.htm.
14. ISO/IEC PDTR 15446, Guide for the production of PP and ST, Draft, Apr 2000.
15. NIAP, CC Toolbox Reference Manual, Version 6.0, 2000.

A Secure Migration Mechanism of Mobile Agents Under Mobile Agent Environments

Dongwon Jeong[1], Young-Gab Kim[2], Young-Shil Kim[3], Lee-Sub Lee[4], Soo-Hyun Park[5], and Doo-Kwon Baik[2]

[1] Dept. of Informatics & Statistics, Kunsan National University
San 68, Miryong-dong, Gunsan, Jeolabuk-do, 573-701, Korea
djeong@kunsan.ac.kr

[2] Department of Computer Science and Engineering , Korea University
Anam-dong 5-ga, Seongbuk-gu, 136-701, Seoul, Korea
{ygkim,baik}@software.korea.ac.kr

[3] Division of Computer Science & Information, Daelim College
Bisan-dong DongAn-ku Anayang-si Kyounggi-do, 431-715 Korea
pewkys@daelim.ac.kr

[4] Dept. of Computer Engineering, Kumoh National Institute of Technology
188, Shinpyung-dong, Gumi, Gyeongbuk
eesub@kumoh.ac.kr

[5] School of Business IT, Kookmin University
861-1, Jeongreung-dong, Sungbuk-ku, SEOUL, 136-701, Korea
shpark21@kookmin.ac.kr

Abstract. Although mobile agent paradigm provides many advantages for distributed computing, several security issues of the mobile agent paradigm remain as the most import difficulty that should be solved to increase its application. One of the most important mobile agent security issues is that mobile agents securely transfer from a host to other hosts. Until now, many mobile agent systems have been developed, but those systems do not support mechanisms to provide for secure transmission of mobile agents or have several problems. This paper proposes an integrity mechanism that mobile agents can migrate to other hosts securely. This mechanism is independent on specific security frameworks, so it can be used easily under various mobile agent environments.

1 Introduction

Mobile agent paradigm has been evolved from transitional client/server paradigm[1,2] and it has many advantages such as mobility, autonomy, intelligence, adaptability, cooperation, etc [3]. Especially, the mobility is the most remarkable property of the mobile agent paradigm. Thus, the security issue of mobile agents and mobile agent systems is recognized as one the most important problems.

The security issues of the mobile agent paradigm are classified into two key issues generally- mobile agent security and mobile agent system security. The security issues of mobile agent systems are related with managing of mobile agent system resource, protecting from malicious agents or malicious hosts, and so on. The mobile agent security services are confidentiality, integrity, authentication, access control, and non-repudiation. Until now, many mobile agent systems have been developed [5,6,7,8,9,10]. These systems provide various security policies and mechanisms to address the issues above.

Mobile agents have mobility. In other words, the mobile agents migrate from a host to other hosts. If a mobile agent can be altered and be a malicious agent when its migration to other hosts, many hosts can be attacked easily. Therefore, the most important security issue is to safely transfer mobile agents from a host to others.

Many mobile agent systems have been tried to solve the aforementioned security issues, but most mobile agent systems provide no mechanism to securely migrate mobile agents to other hosts. Several mobile agent systems support mechanisms to address this issue by a secure transfer protocol. In fact, the secure transfer protocol is different from radical integrity mechanism. In addition, they depend on specific security frameworks, so it causes compatibility problem between the frameworks.

In this paper, we propose an integrity checking mechanism to migrate mobile agent to other hosts securely. The proposed mechanism is based on the bundle authentication mechanism in the OSGi [4]. We devised this mechanism extending the OSGi to be suitable for mobile agent system environments.

The proposed mechanism is independent on existing security frameworks. It also was designed to increase usability. Therefore, this mechanism is lightweight less than other security frameworks and can be used as an integrity secure component.

2 Related Work

This section introduces key security issues of mobile agent paradigm and describes the security policies of the existing mobile agent systems.

2.1 Security Issues in Mobile Agent Paradigm

This section describes security issues of the mobile agent paradigm. The security issues can be classified into two groups- mobile agent security and mobile agent system security. The mobile agent security is to protect mobile agents from attacks of other malicious objects, and the mobile agent system security is to protect mobile agent systems from attacks of other objects. The objects may be mobile agents or mobile agent systems.

Table 1 shows the security issues, i.e., attack types of mobile agent paradigm [11].

As shown in Table 1, there are many attack types against mobile agent systems or mobile agents. In case of mobile agent security, confidentiality, integrity, authentication, access control, and non-repudiation should be basically supported to secure mobile agents from illegal attacks described above.

Mobile agents have several properties- intelligence, autonomy, adaptability, etc. Especially, mobile agents have mobility, and it mobility is the most remarkable property of mobile agent paradigm. Therefore, the integrity issue, secure migration for the mobile agents, is considered as one of the most important security issues. If anonymous mobile agent is illegally alerted while migration, it causes many serious problems. As a result, the integrity of mobile agents should be guarantee preferentially under the mobile agent environments.

Table 1. Attack Types in Mobile Agent and System

Attacker	Target	Attack type
Mobile Agent	Mobile Agent System	- Camouflage
		- Paralysis of mobile agent systems
		- Illegal access to resources or services
Mobile Agent	Mobile Agent	- Camouflage
		- Interfering with other mobile agents
		- Denial of receipt or transmission
		- Degeneration of mobile agents
Mobile Agent System	Mobile Agent	- Camouflage
		- Tapping status or code of MA
		- Altering code or status of MA
Mobile Agent System	Mobile Agent System	- Camouflage
		- Illegal accessing services or data
		- Illegal message copy or response

2.2 Security Policy of Mobile Agent Systems

This section introduces security policies of existing mobile agent systems. Table 2 shows a summary of the security policies supported by the mobile agent systems- Aglet [5], Voyager [6], Concordia [7], Agent Tcl [8], Ajanta [9], and SOMA [10].

Almost all the mobile agent systems provide secure mechanisms for hosts. On the other hands, in case of mobile agent security, only several systems such as SOMA and Ajanta support the security mechanisms for mobile agents.

As shown in table 2, SOMA and Ajanta provide mechanisms for securing agent status and gathered data respectively. Especially, Ajanta uses the PKI, one of the security frameworks, so a certificate authority is required for authentication between hosts. It makes authentication process slow, and it also causes heavy mobile agent systems. It causes compatibility problem between various mobile agent systems because of dependency on specific security frameworks.

This paper focuses on the integrity issue. In other words, the goal of this paper is to develop a mechanism that secures both of status and data of agents and is independent on specific security frameworks.

3 Preconditions and Overall Process

This section predefines several constraints and overall process of the proposed integrity checking mechanism.

3.1 Preconditions

The integrity mechanism proposed in this paper has several preconditions. We already referred that this paper focuses on the integrity issue for mobile agents, so it is necessary for these preconditions to be defined. The summary of these preconditions is as follows:

Table 2. Security Policies of Mobile Agent Systems

System	MAS security	MA security
Aglet	- Access control based on proxy object - Trusted and untrusted	N/A
Voyager	- Extending Java security manager	N/A
Concordia	- Access control based on by security manager - The security manager depend on users' identity	N/A
Agent Tcl	- Access control depending on its security policy - Anonymous and authenticated	N/A
Ajanta	- Access control by the mechanism based on proxy	- Securing agent status
SOMA	- Roll-based access control based on hierarchical security policy	- Integrity of data gathered

All hosts in the enclosed environments safe from any of threats. Mutual authentication between all hosts is accomplished in the initial of mobile agent systems. The hosts authenticated can trust mutually. The authenticated hosts never alter mobile agents migrated from other hosts. Mobile agents have an itinerary that is planned in advance.

As aforementioned, there are many security issues for the mobile agent paradigm, but this paper focuses on the integrity for secure migration of agents. Thus, basic preconditions are required to describe the proposed integrity mechanism.

First, we suppose that the security of mobile agent systems, i.e., security of hosts is guaranteed. This means that all hosts free from any threats and do not altered by malicious other objects. We suppose that all hosts are reliable. Thus, all mobile agents secure from the hosts that they migrate to and stay in.

The mobile agents have mobility. Hence, they have an itinerary that they will visit. Various itinerary distribution policies are possible. In this paper, we suppose all mobile agents have an itinerary including all addresses of the hosts.

3.2 Overall Process of the Integrity Checking Mechanism

The proposed integrity mechanism consists of two main steps- mutual authentication step and secure migration step. The first step is to authenticate mutually between mobile agent systems, hosts. The second step is that mobile agents migrate to and work on other hosts.

Figure 1 depicts the overall process of the proposed integrity checking mechanism. At the first step in this figure, all hosts are authenticated mutually. The given function, h2h() means the mutual authentication is accomplished each other over all hosts. Therefore, if the number of all hosts is N, the whole mutual authentication operation is accomplished as many as a from N=1 to (N-1) N , i.e., N*(N-1)/2. As a result, the number of one-way authentication operations is Factorial [N-1], i.e., (N-1)!.

This paper focuses on the integrity mechanism not the mutual authentication. Thus, this paper defined a simple authentication mechanism and implemented uses using the HmacSHA1 algorithm provided by SUN.

After the mutual authentication, a closed environment is organized. Then, an itinerary is prepared, and then a mobile agent migrates to the hosts with the itinerary. The itinerary may include a subset or all of the hosts in the closed environment.

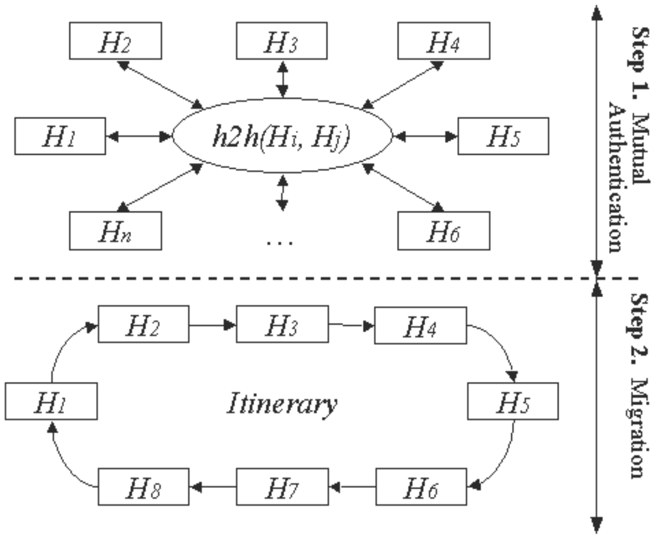


Fig. 1. Main Process of the Integrity Checking Mechanism

4 Integrity Checking Mechanism

This section describes the integrity checking mechanism proposed in this paper.

4.1 Mutual Authentication Phase

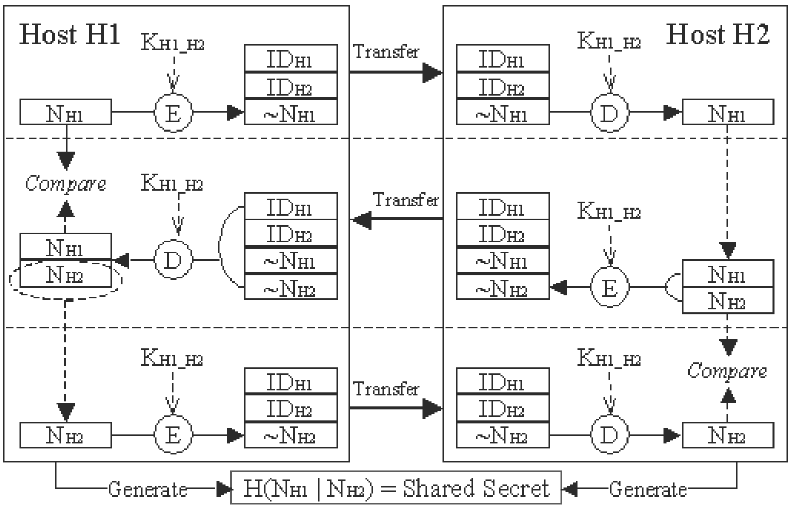
Suppose that anonymous host, H1 sends a mobile agent to H2 for examining computer viruses. Before sending the mobile agent to H2, H1 is willing to check whether H2 is reliable. If H1 sends the mobile agent without reliability examination and H2 is not reliable, then the mobile agent may be changed by H2.

This paper focuses on the secure transfer of mobile agents not the mutual authentication between hosts. However, a mutual authentication is basically required for realizing the secure migration of mobile agents. Thus, this paper designs and uses a simple authentication mechanism. The mutual authentication mechanism is shown in the following figure.

Figure 2 shows the mutual authentication process between two hosts in detail. Host H1 generates a nonce, NH1 using a symmetric key. The generated nonce, N_{H1} is encrypted with the symmetric key, K_{H1_H2}. The encrypted nonce, $\tilde{}N_{H1}$ is transferred to

Host H2. H2 decrypts $\tilde{}N_{H1}$ generated by H1 with K_{H1_H2}. H2 generates a nonce, N_{H2}. Then, H2 encrypts the new nonce and N_{H1} together, and send the result to H1 again. H1 decrypts the result and compares the original N_{H1} and N_{H1} from H2. If the nonce from H2, N_{H1} is different from the original nonce, this mutual authentication is cleaned up. If not, H1 encrypts the nonce N_{H2} and sends to H2 again. H2 also compares the original nonce and the nonce from H1.

When the mutual authentication between them is completed, a shared secret key is generated. Both of hosts create the shared secret key with a same hash function respectively. The shared secret key and two nonce values are used for encryption and decryption of a mobile agent.



K_{H1_H2} : Symmetric key
N_{H1}, N_{H2} : Original nonce values generated H1 and H2 respectively
$\sim N_{H1}$, $\sim N_{H2}$: Nonce values encrypted
ID_{H1}, ID_{H2} : Identifications of H1 and H2

Fig. 2. Mutual Authentication Process between Hosts

The process described above is accomplished iteratively until all hosts authenticate each other. Thus, the mutual authentication operation is required as many as a from N=1 to (N-1) N where the number of all hosts is N.

4.2 Secure Mobile Agent Migration Phase

This section describes the integrity mechanism, the most crucial part of this paper. As above-mentioned, the objective of this integrity mechanism is to allow mobile agents to securely migrate to other hosts. This mechanism uses the shared secret key generated and nonce values generated from the mutual authentication process. Figure 3 shows the integrity mechanism between hosts in detail.

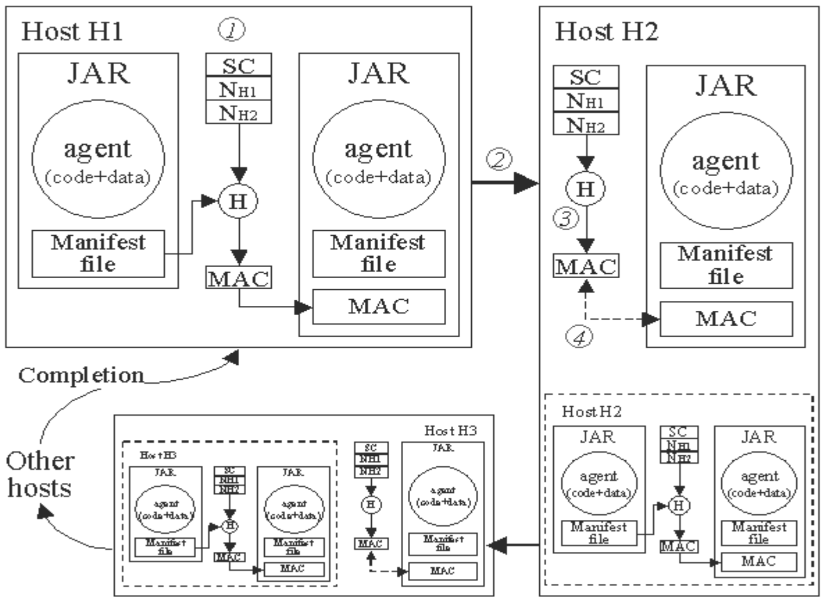


Fig. 3. Secure Migration of Mobile Agents in the Integrity Checking Mechanism

In this figure, the MAC is generated by the encrypt hash function, named HMAC function. This paper uses the HmacSHA1 algorithm provided by SUN. We implemented the proposed mechanism using the hash function. The generated MAC is appended to the signed JAR file and the JAR file is sent to Host B. In other words, the agent on Host A migrates to Host B.

Host B receives the JAR file. Host B also generates a MAC using the same hash function with Host A. Host B gives the shared secret key and the nonce values to the hash function. A MAC is generated through this process. Host B compares the generated MAC with the MAC in the receiving JAR file that is generated on Host A. If the agent was not changed by anonymous malicious object while transmitting, both of the MACs are same. If the JAR file is changed, the original MAC is different from the final MAC.

The sequential processing steps of this mechanism in figure 3 is as follows:

(1)Generating a signed JAR file including the agent to be migrated to Host B. In this step, the manifest file of the original JAR is used as an input of the hash function. The hash function encrypts the manifest file with the shared secret key and two nonce values through the mutual authentication. A MAC is generated as the result of the encryption

(2)Transferring the JAR file (Migration of the agent). The final JAR file on Host A is sent to Host B. It means the agent migrates from Host A to Host B.

(3)Generating a MAC on Host B. Host B generates a MAC. Host B also uses the same hash function, shared secret key, two nonce values, and the manifest file within the JAR file from Host A is used an input.

(4)Comparing of two MACs. The MAC of Host B is compared with the MAC generated and sent on Host A.

4.3 Comparison

This section describes the comparison result of the proposed integrity mechanism and the existing mobile agent systems. Table 3 shows the summary of comparison result.

Table 3. Qualitative Comparison

Comparative Item	SOMA	Ajanta	Proposed Mechanism
Independency	N/A	N/A	Support
Size of MA encrypted	Large	Large	Small
Encryption speed	Slow	Slow	Fast
Transmission speed	Slow	Slow	Fast
Compatibility	Low	Low	High
Application	Low	Low	High

The goal of this paper is to an integrity mechanism for secure migration. Until now, many several mobile agent systems have been developed, but most mobile agent systems do not provide such an integrity mechanism. In case of several mobile agent systems supporting it, they depend on the specific security frameworks. For example, Ajanta is based on the PKI, so its size is large and also requires a certain authenticating organization.

The proposed integrity mechanism does not dependent on the existing security frameworks. Therefore, it is light-weighted and can be integrated into other security architecture. The existing mobile agent systems encrypt the entire agent. It makes the encryption speed slow, and it causes the decrease of migration speed. However, the propose mechanism encrypt the manifest file not the whole. Therefore, this integrity mechanism solves the issues above.

5 Conclusion

In this paper, we proposed an integrity mechanism for secure migration of mobile agents. The mechanism has two key stages- mutual authentication and secure migration. This paper focuses on the secure migration, i.e., the mechanism to provide mobile agents' integrity for their transferring. Hence, we used a simple mutual authentication mechanism to authenticate between hosts.

The goal of this paper is to provide an integrity mechanism for secure migration of mobile agents. To achieve this goal, this paper extended the bundle authentication mechanism in OSGi. The OSGi bundle authentication mechanism is based on the PKI. It has several problems as referred in the section 4.3.

The integrity mechanism proposed in this paper is based on MAC. Also we used the only manifest file not the whole file in aspect of encryption operation. As a result, it has several advantages. The first is that it makes the transmission speed of the agents increase. The second is that the encrypt operation is rapid. Finally, this mechanism does not depend on any other security frameworks. Therefore, the mobile agent systems using this mechanism become light-weighted. In addition, we can integrate this mechanism into other security architectures easily.

References

1. D.B. Lange and M. Oshima, “Programming and deploying Java mobile agents with aglets," Addison-wesley, 1998.
2. M.M. Karnik and A.R. Tripathi, “Design Issues in Mobile Agent Programming Systems," IEEE Concurrency, pp.52-61, July 1998.
3. D.B. Lange and M. Oshima, “Seven Good Reasons for Mobile Agents," Communications of the ACM, vol. 42, no 3, March 1999.
4. OGSi, http://www.osgi.org/
5. G. Karjoth, D.B. Lange, and M. Oshima, “A Security Model for Aglets." Lecture Notes in Computer Science, vol. LNCS 1419, pp.188-205, 1998.
6. Voyager, http://www.recursionsw.com/products/voyager
7. T. Walsho, No. Pariorek, and D. Wong, “Security and Reliability in Corcordia," The 31st Annual Hawaii International Conference on System Sciences, Hawaii, 1998.
8. R.S. Gray and et al., “Agent: Security in a multiple-language, mobile-agent system," Lecture Notes in Computer Science, vol. LNCS 1419, 1998.
9. N. Karnik, A. Tripathi, “Security in the Ajanta Mobile Agent System," Software- Practice and Experience, January 2001.
10. A. Corradi, R. Montanari, and C. Stefanelli, “Mobile Agents Integrity in Ecommerce Applications," Proceedings of the 19th IEEE ICDCS’99, Austin, Texas, May 1999. T.A. Jones, “Writing a good paper," IEEE Trans. on General Writing, vol. 1, no. 2, pp.1-10, May 2002.
11. W. Jansen and T. Karygiannis, “Mobile Agent Security," NIST, Special Publication 800-19, August 1999.

A Flexible Privilege Management Scheme for Role Graph Model

Yuna Jung[1] and Eenjun Hwang[2,⋆]

[1] The Graduate School of Information and Communication, Ajou University
Suwon, Korea, 442-749
serazade@ajou.ac.kr

[2] Department of Electronics and Computer Engineering, Korea University
Seoul, Korea, 136-701
Tel: +82-2-3290-3256
ehwang04@korea.ac.kr

Abstract. Since the role-based access control was introduced in the early 1970s, it has been considered as one of the promising access control methods. The role graph model was suggested as a reference model for the role-based access control. However, its privilege management is too strict to be applied to various applications. In this paper, therefore, we propose a flexible privilege management scheme based on the refinement of privileges in the role graph model, and show its effectiveness through several scenarios. We expect that this scheme will make the role graph model more powerful and applicable.

1 Introduction

The Role-based Access Control (*RBAC*) model [1] has emerged since 1990s as a promising approach for managing and enforcing security in a huge and complex system. The essential idea of *RBAC* is that privileges are associated with roles, and users can have privileges by being assigned to appropriate roles. The role is a major component in constructing a role-based access control model. In *RBAC* model, system administrators can create roles, grant privileges to those roles, and then assign users to the roles on the basis of their specific job responsibility and organization policy. During this step, role-privilege relationships are defined, which makes it very simple to assign users to the defined roles. Without RBAC, it would be difficult to determine what privileges have been authorized to which users. Generally, assigning users to roles requires less technical skill than assigning privileges to roles. Besides, role-based approach is more realistic than user-based approach since the structure of access right is based on roles in most organizations. Due to this property, *RBAC* has become the most popular access control model. *RBAC* model has evolved from $RBAC_0$ to $RBAC_3$. As a fundamental stage, $RBAC_0$ incorporated the minimal requirements of *RBAC* system. $RBAC_1$ added role hierarchies into $RBAC_0$, while $RBAC_2$ added constraints to $RBAC_0$. $RBAC_3$ combined $RBAC_1$ and $RBAC_2$. The role graph model, which was introduced by Nyanchama and Osborn, is a reference model for role-based access control, and provides a

⋆ Corresponding Author.

way of visualizing interactions among roles. In 1996, algorithms for manipulating role graphs were also implemented by them [3]. These algorithms allow roles and privileges to be created, deleted, and altered. They have improved the role graph model and provided a basis for implementing and maintaining *RBAC*.

So far, most researches have focused on the role engineering, such as constructing and maintaining role hierarchy, assigning users to roles, and so on. However, in *RBAC* model, privilege changes are more frequent than role changes. Objects of a privilege are continuously created, deleted, and modified. Even more, privileges are also changing dynamically. However, in spite of the importance of privilege management, existing management are too simple and strict. Therefore, it is necessary to improve the privilege engineering to make *RBAC* model more practical. In this paper, we propose a privilege refinement scheme to achieve such privilege engineering.

The rest of the paper is organized as follows. In Section 2, we first give a brief introduction to the role graph model and then introduce how to handle privilege conflicts in the existing role graph model. In Section 3, we propose a scheme to extend the role graph model through refining privileges. Then, in Section 4, we conclude the paper and discuss some of future works.

2 Role Graph Model

The role graph model consists of three distinct entities; privileges, roles, and users. A privilege denotes an access authorization for an object and is described by two elements. One of them is a set of objects such as system resources or data, and the other is an operation for the objects. A role is a named set of privileges and is represented by a role name and a set of privileges. Role-role relationships are based on subset relation between roles and are represented by *is-junior* relationship on the role hierarchy [4]. A user is the subject of access control models, and wants to access system objects. To efficiently manage a number of users who should have the same authorization, the role graph model defines a *group* for a set of users. This is very similar to the user group in the UNIX operating system.

The role graph is, in essence, an acyclic, directed graph in which a node represents a role in the system, and an edge represents a *is-junior* relationship [2]. Every role graph has one *MaxRole* and one *MinRole*. The *MaxRole* is a common senior of all the roles in the role graph. According to the *is-junior* relationship, it has all the privileges of the roles in the role graph. The *MinRole* is a common junior of all the roles in the role graph and represents the minimum set of privileges available to all the roles. According to the *is-junior* relationship, privileges of the *MinRole* are inherited to all the roles in the graph. However, *MaxRole* and *MinRole* need not necessarily be assigned to any user, because it could exist as a conceptual role. In the role graph model, senior roles have all privileges of their junior roles. Therefore, actual privileges of one role consist of two types: its own privileges called direct privileges, and the privileges inherited from its juniors. A set of actual privileges of a role is called as its *effective privilege*. In Fig. 1 and Fig. 2, p_is in the bracket represent direct privileges of a role.

2.1 Handling Privilege-Privilege Conflicts on the Role Graph Model

To apply the role-based access control model to real applications, we should handle conflicts of interest efficiently [5]. Negligent manipulation of such conflicts might cause serious security threat. For example, in a complex environment where the actions of an ill-intentioned user can cause harmful damages, granting a user certain combination of privileges that could be a threat should be prevented.

Due to such threat, it is very important in the role graph model to deal with conflicts of interest. Nyanchama and Osborn [3] discussed conflicts of interest on the role graph model in 1997. First, they defined five different kinds of conflicts, which could arise in the role graph model; user-user/group-group/user-group conflict, role-role conflict, privilege-privilege conflict, user-role assignment conflict, and role-privilege assignment conflict. Then, they suggested algorithms for preventing two kinds of conflicts, role-role conflict and privilege-privilege conflict. A role-role conflict means that a user/group has two roles that should not appear together. A privilege-privilege conflict means that a role has two privileges that should not appear together.

To manipulate privilege-privilege conflicts, they defined *P-Conflicts*, a set of pairs of conflicting privileges. Then, they ensured that no role (except for *MaxRole*) could have two privileges that were defined as a conflicting pair in the *P-Conflicts*. For the integrity, *P-Conflicts* must be referred whenever a system administrator performs maintenance operations of the role graph model such as adding a privilege to an existing role, creating a new role, or inserting an edge between roles. If such an operation would cause a conflicting pair in the effective privilege set of any role, then that operation is aborted.

2.2 Motivation

Strict privilege manipulation of current role graph model might cause serious problems. First of all, the role graph could be very complex since privileges and roles are created and modified even though the modification is trivial. Moreover, such modification can be rejected if a conflict occurs. However, usually, a actual conflicting portion of the most conflicting pairs is partial. Nevertheless, the existing role graph model strictly aborts all modifications that caused a conflict. Such handling obstructs the *RBAC* model from being practical. Let's examine two cases to see how to manipulate privileges in the current role graph model.

Fig.1 (a) shows the case where a new object is added to the object set of p_2. Here, we assume that *L2* and *L4* roles can access new object of p_2, but *L3* cannot. In this case, we should create a new role, which contains a set of expanded objects of p_2, and should cut edges between *L2* and *S2*, and between *L4* and *S2*. That is, we might create a new role graph whenever privileges are modified. Actually, privileges can be changed dynamically in the real environment. Also, role objects are changed frequently. Summarizing the situation, a role graph is continuously changed, and that makes a role graph more complex.

Fig.1 (b) shows the case where a conflict arises between p_{11} and p_{14}, when a new privilege p_{14} is inserted into *L4*. In the current role graph model, insertion operation is aborted by a conflict even though the conflicting part between p_{11} and p_{14} is very small. Consequently, such simple and strict handling of privilege conflicts is not practical at all.

3 Managing Privileges on the Role Graph Model

With strict privilege handling, it is very difficult to apply the role graph model to various applications. To relieve the situation, conflict handling method should be modified in

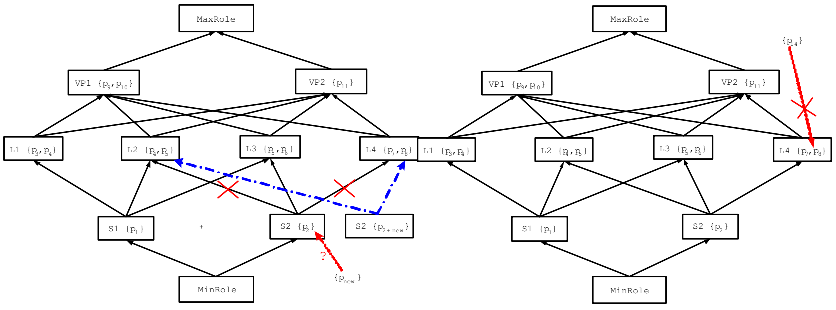


Fig. 1. (a) Adding a new object to privilege p_2, (b) Inserting a new privilege into role *L4*

such a way to give flexibility. In this paper, we propose a flexible privilege management scheme using privilege refinement on the role graph model.

3.1 Privilege Refinement Scheme

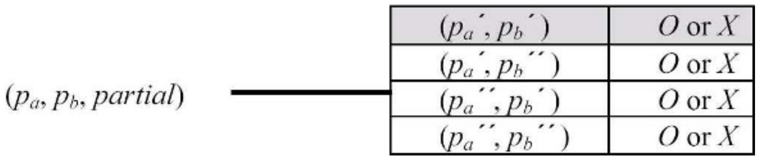
When a privilege-privilege conflict arises due to some privilege change, the operation is abandoned in the current role graph model. In this paper, we are improving the privilege conflict management by refining privileges. For the privilege refinement, a pair of two conflicting privileges in *P-Conflicts* are divided into two groups. One of them contains those conflicting pairs only and the other group contains all the remaining pairs. Then, the latter group can exist in a role graph without any trouble. For the flexibility of privilege management, we modified the role graph model as follows. With several exceptions related to privilege refinement, we stick to the definition of the current role graph model.

A set $O = \{o_1, \ldots, o_v\}$ is a set of objects such as system resources or data that users want to access. $X_i = \{x_i | x_i : \text{object of } p_i, x_i \in O\}$ is a set of objects belonging to p_i, and each element of x_i is an element of *O*. Another set $P = \{P_1, \ldots, P_n\}$ is a set of all privileges on a role graph, and *P* is the same as *effective privilege* of *MaxRole*. A privilege p_i is represented by a pair (X_i, m), where X_i is a set of objects and *m* is the operation. $E = \{e_1, e_2, e_3, \ldots, e_l\}$ is a set of edges connecting one role to another, and each edge represents the *is-junior* relationship between two roles. We can say that role r_a *is-junior* to r_b, iff $r_a.rpset \subset r_b.rpset$. An edge $e_l = (r_a, r_b)$ is represented by an ordered pair (r_a, r_b), where the first element r_a *is-junior* to r_b and second element r_b *is-senior* to r_a. A set of roles $R = \{r_1, r_2, r_3, \ldots, r_m\}$ is a set of all roles in a role graph, where $r_j = (rname, rpset)$ is a named set of privileges. Here, *rname* is the name of the role and *rpset* represents a set of privileges of the role. *rpset*, the *effective privilege* of the role, is union of *direct privilege* of the role and *inherited privilege* from all juniors of the role.

(Definition 1) A role graph $G = \{R, E\}$ is an acyclic graph, where *R* is a set of roles and *E* is a set of edges between roles. *G* has one *MaxRole* and one *MinRole*. A privilege p_n can be refined into two privileges $p_{n'}$ and $p_{n''}$ by separating its objects, where $p_{n'} = (X_{i'}, m)$, $X_{i'} = \{x_{i'} | x_{i'}$: a subset of x_i, $x_i \in O\}$ and $p_{n''} = (X_{i''}, m)$,

$X_i'' = \{x_i'' | x_i'' = x_i - x_i', x_i \in O\}$.

There are two kinds of privilege-privilege conflicts. One is *full privilege-privilege conflict*, and other is *partial privilege-privilege conflict*. *Full privilege-privilege conflict* cannot be fragmented, and should be handled strictly. (p_a, p_b, *full*) represents the *full privilege-privilege conflict* between p_a and p_b. On the other hand, *partial privilege-privilege conflict* can be fragmented into problematic part and the rest. (p_a, p_b, *partial*) represents the *partial privilege-privilege conflict*. Except for conflicting part of (p_a, p_b), the rest part can exist in the role graph. Each *partial privilege-privilege conflict* is classified again into four types as follows.


(p_a, p_b, *partial*)

(p_a', p_b')	O or X
(p_a', p_b'')	O or X
(p_a'', p_b')	O or X
(p_a'', p_b'')	O or X

A partial conflict pair is to be separated into four subsets, (p_a', p_b'), (p_a', p_b''), (p_a'', p_b'), and (p_a'', p_b''). Here, p_a' is a problematic part of p_a, which bring on a conflict between p_a and p_b, and p_b' is a problematic part of p_b. First of all, we fragment a privilege into problematic part and the rest, and make four combinations using four privilege pieces (Each privilege is divided into two parts). In the table of a fragmented conflict pair, possibility for inserting is represented as *O* (approval) or *X* (disapproval). After all, we can insert an authorized portion of a partial conflicting privilege even if the portion is just partial. However, it is in fact impossible to insert (p_a', p_b') because both p_a' and p_b' are just problematic parts that caused the conflict. Therefore, we use three subset of a conflict pair except for (p_a', p_b'). Here, a security administrator defines p_a' and p_b' directly. If the problematic part is on both sides, p_a and p_a, only p_a'' and p_b'' can coexist in a role graph. If the problematic part is on one side, p_a or p_b, then either (p_a, p_b'') or (p_a'', p_b), can exist by separating a problematic privilege. When (p_a', p_b'') and (p_a'', p_b'') are marked as *O* (approval), (p_a, p_b'') can be accepted. On the other side, (p_a'', p_b') and (p_a'', p_b'') are marked as *O* (approval), (p_a'', p_b) can be accepted. As you see, we can improve flexibility of privilege insertion using the partial insertion of conflicting privilege.

(Definition 2) $P{-}Conflict = \{(p_a, p_b, full), (p_c, p_d, partial), \ldots, (p_i, p_j, partial)\}$ is a set of pairs of conflicting privileges, and is composed of *full privilege-privilege conflict* pairs and *partial privilege-privilege conflict* pairs.

3.2 Flexible Privilege Management Using the Privilege Refinement

As we mentioned, existing privilege management ignores the real situation. Therefore, in this paper, we propose more bendable handling of privileges using privilege refinement. In Fig.2 (a) and (b), we show how we improve the privilege management in the role graph using the privilege refinement.

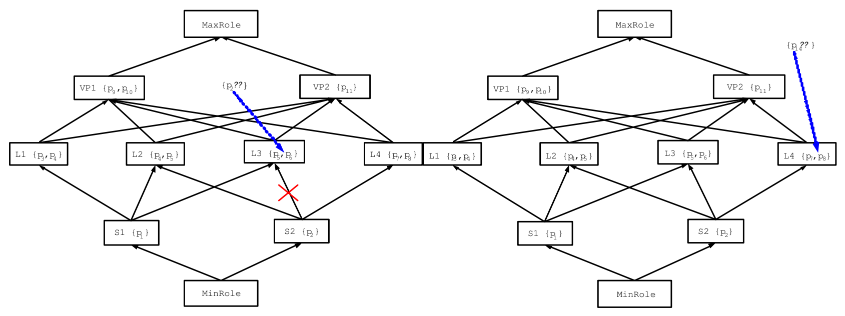


Fig. 2. (a) Privilege refinement for Fig.1 (a), (b) Privilege refinement for Fig.1 (b)

4 Scenario

In this section, we will consider four scenarios where our proposed scheme can support flexible insertion. To make the situation clearer, we use the role graph in Fig. 1. as a sample role graph. In addition, we assume the following P-Conflicts.
P-Conflicts = $\{(P_{10}, P_{11}, full), (P_6, P_{12}, partial), (P_4, P_{13}, partial), (P_1, P_{14}, partial)\}$

4.1 Scenario 1: Handling a Full Privilege Conflict

Let's add p_{10} to *VP2* as a *direct privilege* of *VP2*. However, p_{10} and p_{11} are fully conflicting each other. In case of the full conflict, there is no space for compromise because each object of two privileges collides against the other. Therefore, we manage such case with a strict method used in the existing role graph model, so the operation is denied. This process is shown in Fig. 3.

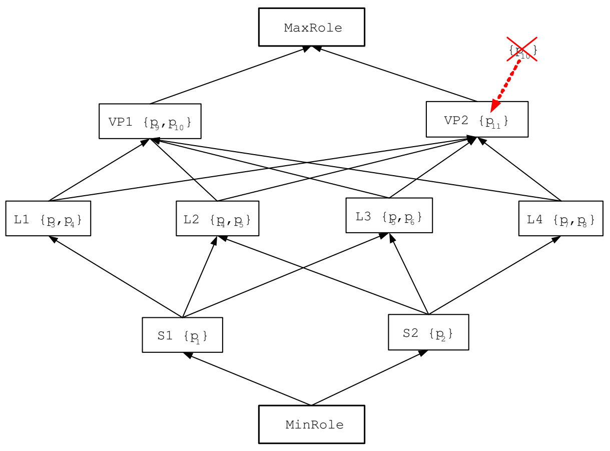
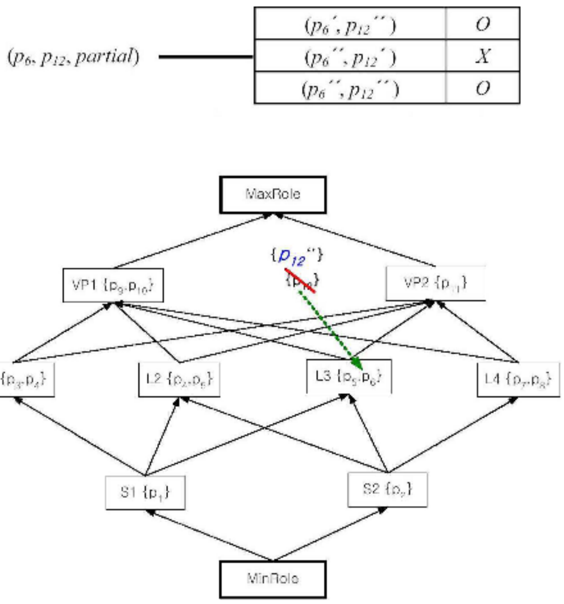


Fig. 3. Handling a full privilege conflicts

4.2 Scenario 2: Modifying an Inserted Privilege

Let's assume that we try to add p_{12} to *L3*. However, p_{12} is conflicting with p_{11} at *effective privilege* of *VP2*. Fortunately, the conflict is a partial conflict (You can find out a type of the conflict from its notation). In case of partial conflict, we can insert a part of p_{12} by separating the problematic part from the privilege. We assume that a table of conflict refinement is as follows. Authorized forms are ($p_6\prime$, $p_{12}\prime\prime$) and ($p_6\prime\prime$, $p_{12}\prime\prime$). It means that (p_6 , $p_{12}\prime\prime$) do not cause a trouble, so the conflict pair should be modified into (p_6 , $p_{12}\prime\prime$) to be inserted to a role graph. Therefore, $p_{12}\prime\prime$ is inserted instead of p_{12}. This process is shown in Fig. 4.



(p_6', p_{12}'')	*O*
(p_6'', p_{12}')	*X*
(p_6'', p_{12}'')	*O*

Fig. 4. An example of modifying an inserted privilege

4.3 Scenario 3: Modifying an Existing Direct Privilege

We want to add p_{13} to *L2*, but p_{13} is conflicting with p_4, an existing privilege of *L2*. However, we can insert a part of conflicting privilege because this conflict is partial. Let's assume that a table of conflict refinement is as follows. According to this, $p_4\prime\prime$ and p_{13} can coexist without any conflict, because ($p_4\prime\prime$, $p_{13}\prime$) and ($p_4\prime\prime$, $p_{13}\prime\prime$) are marked with *O*. Therefore, $p_4\prime\prime$ replaces p_4 as a *direct privilege* of *L2*, and then p_{13} is inserted. This process is shown in Fig. 5.

4.4 Scenario 4: Modifying an Existing Inherited Privilege

Let's suppose that we want to add p_{14} to *VP1*. As you know, however, p_{14} is conflicting with p_1, an existing privilege of *VP1*. We can insert a part of the conflicting privilege

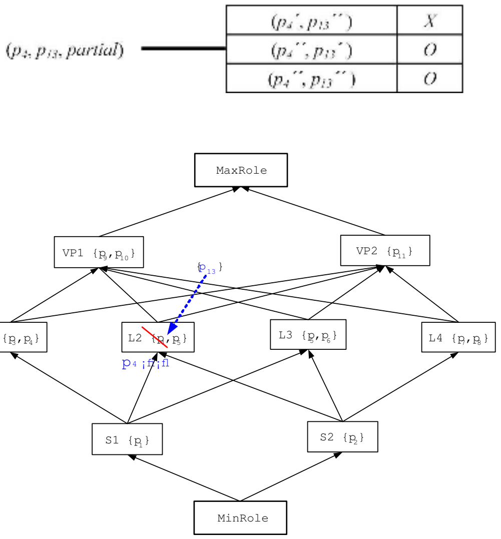


Fig. 5. An example of modifying an existing directed privilege

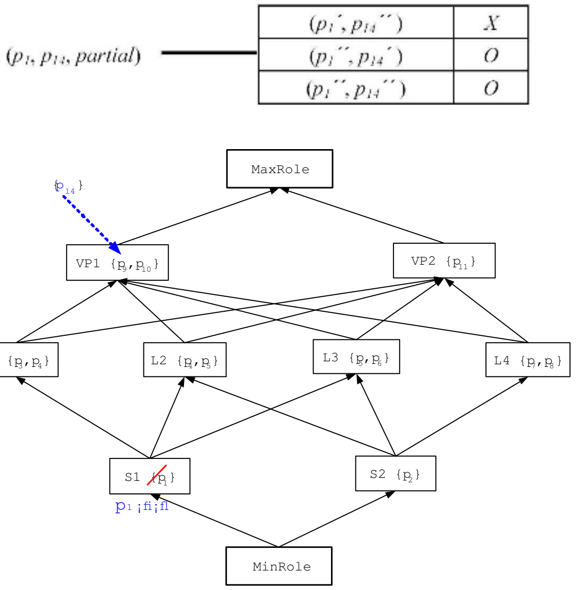


Fig. 6. An example of modifying an existing inherited privilege

because this conflict is partial. However, p_1 is an *inherited privilege* from *S1*. If a table of conflict refinement is as follows, authorized form is (p_1'', p_{14}), because (p_1'', p_{14}') and (p_1'', p_{14}'') are approved. Therefore, we should change a conflicting pair, (p_1, p_{14}), into a non-conflicting pair, (p_1'', p_{14}). Therefore, p_1'' replaces p_1 as a *direct privilege* of *S1*, and then p_{14} can be inserted into *VP1*. This process is shown in Fig. 6.

5 Conclusion

Since the role-based access control model was introduced in the early 1990s, it has been applied to various security systems as a promising access control method. However, its privilege conflict management is too strict to be adapted to real applications. In this paper, we proposed a more flexible handling scheme for privilege conflicts using privilege refinement, and examined various cases to show its effectiveness.

Acknowledgements

This research has been supported by University IT Research Center Project. We would like to thank Prof. Hongjin Yeh for his valuable comments.

References

1. R.S. Sandhu, "Role-Based Access Control", IEEE Computer, pp. 38-47, Feb., (1996).
2. Nyanchama, M., Osborn, S. L., Access rights administration in role-based security systems. In Proc. of the IFIP Working Group 11.3, Working Conference on Database Security, Elsevier North-Holland, Amsterdam, The Netherlands, (1994).
3. Nyanchama, M., Osborn, S. L., The Role Graph Model and Conflict of Interest, ACM transactions on Information and System Security, Vol. 2, No. 1, Feb. (1999).
4. R.S. Sandhu et.al, Role Hierarchies and Constraints for Lattice-based access controls, In Proc. of the Conf. On Computer Security, springer-Verlag, New York, (1996).
5. Simon R., Zurko M. E., Separation of duty in role based access control environments, In Proc. of the 10th IEEE Computer Society Press, Los Alamitos, CA, (1997).
6. Matunda Nyanchama and Sylvia Osborn, "Access rights administration in role-based access control revisited", In Proceedings of the IEIP Working Group 11.3 Working Conference on Database Security,Amsterdam, The Netherlands,(1994).
7. Matunda Nyanchama and Sylvia Osborn, "The Role Graph Model and Conflict of Interest", ACM Transactions on Information and System Security, Vol.2, No.1, pp. 3-33, Feb., (1999).

The System Modeling for Detections of New Malicious Codes

EunYoung Kim, CheolHo Lee, HyungGeun Oh, and JinSeok Lee

National Security Research Institute
62-1 Hwa-am-dong, Yu-seong-gu
Daejeon, 305-718, Republic of Korea
{eykim,chlee,hgoh,jinslee}@etri.re.kr

Abstract. During the last several years, Malicious code have become increasingly more sophisticated. At the same time, the Internet's growing popularity and the steady adoption of broadband technologies have allowed malicious codes to spread quickly. However, traditional anti-malicious codes detections's method is pattern matching. Pattern matching's method just can detect within the narrow limits of known malicious codes. That is, in the past, pattern matching method was able to ship new pattern for most malicious codes before they could achieve widespread distribution. If malicious code software vendors could not provide new pattern, nobody can not detect new malicious code. Accordingly, users were hacked by somebody hacker. In this article, we suggest the new malicious code detection algorithm and the system modelings without malicious pattern DB.

Keywords: Malicious Code, Realtime monitoring.

1 Introduction

As the popularity of Internet and computer increases continuously along with the fast development of IT, the number of computer crimes increases dramatically. Specially malicious codes take advantage of various security holes of computer systems, and these codes are easily and widely distributed through the Internet. These malicious codes threaten core infrastructures of a country because the number of information hacking incidents at colleges, research institutes, and governmental organizations increases annually, and so do damage costs. In addition, hacking techniques and malicious development techniques continue to advance, but the malicious code detection techniques cannot catch up with the advance of hacking techniques.

Statistics of computer incidents between 2002 and 2003 in Korea are shown in [Fig. 1]. These statistical data is published by CERTCC-KR in Korea [1]. The occurrences of malicious code hacking grew rapidly after 2001, and they continue to grow every year. Because the number of hacking incidents increases, system administrators encourage users to install personal firewall in their PC.

Personal firewall is bundled with other security products, such as ZoneAlarm, BlackICE, Tiny and so on, and some firewall is available freely to users. Therefore when some data is going to be transmitted from a PC without the knowledge of its user, a personal firewall can check data transmission and may block the transmission. Even

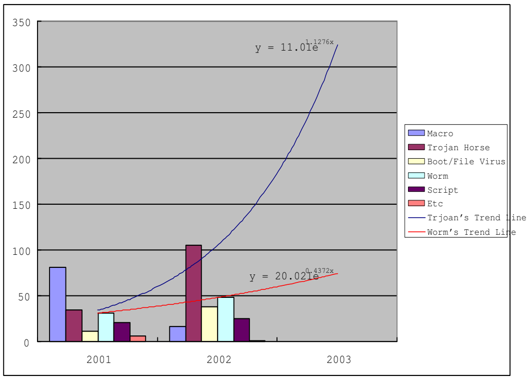


Fig. 1. The Number of Hacking Incidents in South Korea

though personal firewall can detect hacking activities that use network transmissions, they may not be able to detect hacking activities that use malicious codes.

In this article, I would like to suggest a malicious code detection system which consisted of three modules. Three modules are real time System monitoring module, process profiling module and malicious code detection module. hence, The first chapter describes related work which various malicious code's detection method. The second chapter presents the architecture of suggested malicious code detection system. The third chapter, we describe experimental setting and experimental results. and last chapter, we summarize our conclusions and further research issues in section 4th.

2 Related Work

This chapter describes various the malicious code detection methods. There are four kinds of methods: Pattern Matching, Heuristic Method, Policy-based behavior Blocking Method and Expert-based behavior blocking Method [8].

Traditional pattern matching based anti-virus or anti-malicious software detects malicious code by searching for tens of thousands of digital patterns in all scanned files, disks and network status. Each Pattern is a short sequence of bytes extracted from the body of a specific virus strain. If a given pattern is found, the content is reported as infected. However, anti-virus patterns are based on known sequences of bytes from known infections, this technique often fails to detect new virus.

On the contrary to pattern matching technique, heuristic anti-virus technology detects infections by scrutinizing a program's overall structure, its computer instructions and other data contained in the file. The heuristic scanner then makes an assessment of the likelihood that the generally suspicious logic rather than looking for specific patterns.

Behavior blocking software integrates with the operating system of a host computer and monitors program behavior in real-time for malicious actions. The Behavior blocking software then blocks potentially malicious action before they have a chance to affect the system. Monitored behaviors can include:

– Attempts to open, view, delete, and/or modify files
– Attempts to format disk drives and other unrecoverable disk operations
– Modifications to the logic of executable files, scripts of macros
– Modification of critical system settings, such ac start-IP settings
– Scripting of e-mail and instant messaging clients to send executable content
– Initiation of network communications

If the behavior blocker detects that a program a initiating would-be malicious behaviors ac it runs, it can block these behaviors in real-time the offending software. This gives it a fundamental advantage over such established anti-virus detection techniques such as pattern or heuristics. Existing behavior blocking systems can be split into tow categories: policy-based blocking systems and expert-based blocking systems. Policy-based systems allow the administrator to specify an explicit blocking policy stating which behaviors are allowed and which are blocked, Each time a program makes a request to the operating system, the behavior blocker intercepts the request, consults its policy database and either allows the request to proceed, or blocks the request entirely. policy-based behavior blocking system for Java might provide the following options[Table 1]:

Table 1. Policy-based behavior blocking System

Operation description	Block Request
Allows applets to open files	Yes
Allows applets to delete files	Yes
Allows applets to initiate network connections	No
Allows applets to access files in the system directory	No

In contrast to policy-based systems, expert-based systems employ a more opaque method of operation. In these systems, human experts have analyzed entire classes of malicious code and then designed their behavior blocking systems to recognize and block suspicious behaviors. Under some circumstances a would-be dangerous behavior is allowed, and under others, it is blocked. For example, a behavior blocking expert may know the 80% of malicious code first attempts to modify the startup area of the registry before accessing system files. So he can design his behavior blocking system to only block a program's access to system files after first seeing it modify the startup area of the registry. Such a rule is less likely to block legitimate programs and generate false alarms, yet still blocks a high percentage of threats. While a policy-based system might offer an option to "block access to system files" the expert-based system would offer the option to "lock virus-like behavior". Clearly, with such a system, the administrator must take a leap of faith that the experts that built the system have chosen their blocking criteria wisely.

3 The Architecture of Malicious Code's Detection System

The malicious code detection system proposed in this paper consists of three modules, realtime system monitoring module, process profiling module and malicious code detection module[Fig. 2]. The Real-Time System Monitoring module, processes' log data

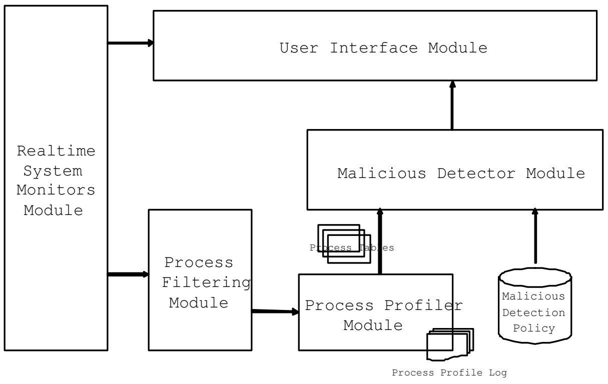


Fig. 2. The Malicious Code's Detection System

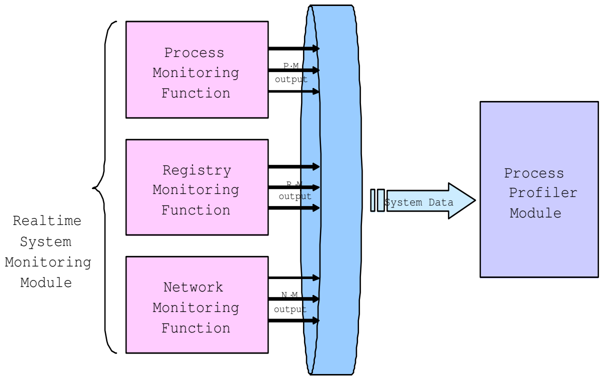

Process Monitoring Function
P-M output
Registry Monitoring Function
R-M output
Network Monitoring Function
N-M output
Realtime System Monitoring Module
System Data
Process Profiler Module

Fig. 3. Real Time System Monitoring Module's Architecture

are transmitted to the process profiling module. The process profiling module re-creates process log data per process. The process profiling module lists information data that is transmitted through process monitoring and network monitoring. Therefore we can monitor user systems if this process profiling works. The malicious code detection module detects malicious activities using the malicious code detection policies. The results of a malicious code detection process and a realtime system monitoring process are informed to a user through user GUI(Graphic User Interface). I will describe each module in detail in the following sections.

The realtime system monitoring module is composed of three submodules: the process monitoring module, the registry monitoring module and the network monitoring module[Fig. 3]. The process monitoring module collects various kinds of information about processes that are executed in user area. The registry monitoring module monitors registry information that all processes approaches. The registry monitoring module is implemented using a registry hooking driver. The network monitoring module gathers network state information, such as which IP addresses and ports are connected. The

Table 2. Output data of the realtime system monitoring module

	Process monitoring	Registry monitoring	Network monitoring
Output Data	– Process list and ID – Process Thread number – Process create time – CPU Utility time of process and Total CPU Utility time – Process DLL list	– Registry path – Registry request type – Registry access time – Registry access result	– Source Address and Port – Destination Address and Port – Port connection state

network monitoring module gets information of network using LSP(Layered Service Protocol). Output data of the realtime system monitoring module is as follows[Table 2].

The process profiling module records profiles of each process based on the logs that are created by the realtime system monitoring module [9]. The process profiling algorithm is as follows. First, compose by queue and store each process' realtime system monitoring data. Because our realtime system monitoring has three monitoring modules, three queues are created for each process. Second, process monitoring information is stored sequentially at each queue in the order of occurrence in the system log. This time work to be process profiling executes depending on each event occurrence smallest time[Fig. 4].

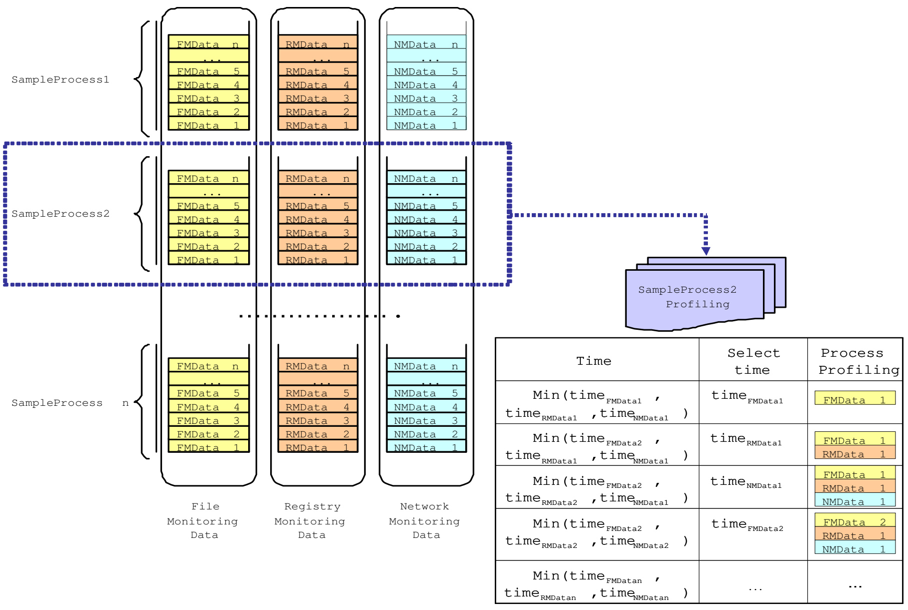
$$ProfilingSelectTime = Min(Time_{FMData}, Time_{RMData}, Time_{NMData}) \tag{3.1}$$

We do this process profiling task so that we can monitor system states and actions of each process. If the module detects detection policies of malicious codes, detection is possible to known malicious codes as well as unknown malicious codes. In addition, we find unknown malicious actions using malicious code profiles.

Our proposed system detects malicious codes in the malicious code detection module using information in a table that is created by the process profiling module. If some process does malicious actions, the module changes the state of relevant process as 'Normal Status $\rightarrow$ Monitoring Status'. If a process with 'Monitoring Status' continues malicious actions and the score of the process exceeds a dangerous point, the state of the process is changed to 'Critical Status'.

However, if the process with 'Monitoring Status' did not do suspicious actions, the process profiling module does not record profiles of the process. State cycles of a process are as follows[Fig. 5].

Malicious code detection module on the basis of malicious action of process malicious code through 3 steps status change of process detection. These reason is false-positive decrease expectation of malicious code detection system. The malicious code detection engine can detect and remove malicious codes according to process status. Actual hacking simulation of the our proposed malicious code detection system has been done.



Time	Select time	Process Profiling
Min($time_{FMData1}$, $time_{RMData1}$, $time_{NMData1}$)	$time_{FMData1}$	FMData 1
Min($time_{FMData2}$, $time_{RMData1}$, $time_{NMData1}$)	$time_{RMData1}$	FMData 1 RMData 1
Min($time_{FMData2}$, $time_{RMData2}$, $time_{NMData1}$)	$time_{NMData1}$	FMData 1 RMData 1 NMData 1
Min($time_{FMData2}$, $time_{RMData2}$, $time_{NMData2}$)	$time_{FMData2}$	FMData 2 RMData 1 NMData 1
Min($time_{FMDatan}$, $time_{RMDatan}$, $time_{NMDatan}$)	...	...

Fig. 4. Process Profiling Algorithm

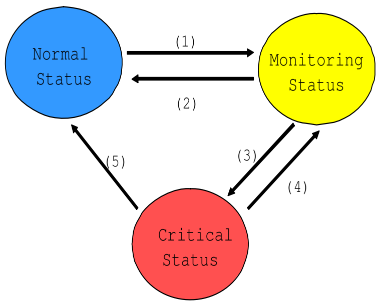


Fig. 5. Process Status Change

4 Experiments

We have implemented a simulated system. In the simulated, the malicious codes that are used in the experiments are malicious codes. Malicious Codes can be classified into two main groups: know Trojan Horses and our developed malicious codes. Malicious Code data were gathered from 200 trojan horse in the Internet. A more elaborate example is Netbus, Schoolbus, Back Orifice in know trojan horse. and then developed malicious codes's character are similar to the know trojan horses.

We construct the simulated system environment on Windows 2000 Professional machines(CPU Pentium 1.4GHz, 256MB Memory) which consist of attacking and victim. Attacking system operated the role of hacker[Fig. 6].

Table 3. Process status change condition

State Number	State Change Condition
1	1. When is process monitoring (a) In case file name that is registered by malicious code conforms (b) In case refer DLL reference that is used in malicious code (c) In case agree with specification information and malicious code database that can get in process monitoring 2. When is registry monitoring (a) In case approach specification registry that approach in malicious code 3. When is network monitoring (a) In case connect specification port that use in malicious code (b) Although specification port that use in malicious code is not, case that port and admitted program that is admitted by user is used normal program
2	When is seceded in 'Monitoring Status' condition
3	State Number 1 'Monitoring Status' reconsideration condition satisfaction
4	When is seceded in 'Critical Status' condition
5	1. Process of in case is disappeared 2. When is exited by user

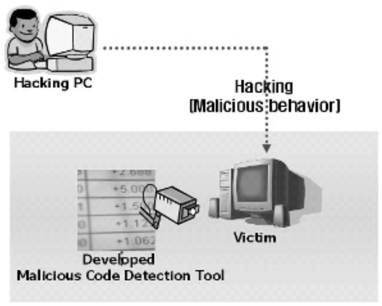

Hacking PC
Hacking [Malicious behavior]
Victim
Developed Malicious Code Detection Tool

Fig. 6. Experiments architecture

The following results were obtained. We experiment on trojan horse with suggested malicious code detection system, got the result 93% detection [3]. It was found from the result that our suggested malicious code detection system could detect the new malicious code without malicious code pattern DB. These results lead us to the conclusion that new malicious codes can detect malicious detection policy [2].

5 Conclusions and Future Directions

In this paper, we have presented the malicious code detection system concentration in real time system monitoring. From what has been discussed above, we can conclude that our suggested malicious code detection system can detect the new malicious code without malicious code pattern DB. As it turned out we got the 93% detection rate. Also,a continuous examination of the mechanism of new malicious code detection algorithm would strengthen this proposition. A further direction of this study will be to provide more efficient detection policy using Machine Learning.

References

1. Korea CERT Coordination Center http://www.krcert.or.kr/
2. H.Han, X.L. Lu, J.Lu, C.Bo, R.L.Yong. Data mining aided signature discovery in network-based intrusion detection system. *ACM SIGOPS Operating Systems Review,* Volume 36 Issue 4, October 2002.
3. PC Flank Experts. The Best Tools to neutralize Trojan horses. http://www.pcflank.com/art17d.htm/. PC Flank Experts team, USA.
4. G.McGraw, G.Morrisett. Attacking Malicious Code. *A Report to the Infosec Research Council*, 2000.
5. M.Roesch. Lightweight intrusion detection for networks. *USENIX In prodcedings of the Thirteenth Systems Administration Conference*, Seattle, WA, USA, 1999.
6. M.Schmall. Heuristic Techniques in AV Solutions: An Overview. http://online.securityfocus.com/infocus/1542. SecurityFocus, February, 2002.
7. M.Schmall. Building an Anti-Virus engine. http://online.securityfocus.com/infocus/1552. SecurityFocus, February, 2002.
8. C.Nachenberg. Behavior Blocking: The Next Step in Anti-Virus Protection. http://online.securityfocus.com/infocus/1557. SecurityFocus, March, 2002.
9. T.F.Lunt. Automated Audit Trail Analysis and Intrusion Detection: A Survey. *11th National Computer Security Conference*, Oct, 1998.

Information Hiding Method Using CDMA on Wave Files

Young-Shil Kim[1], Sang Yun Park[1], Suk-Hee Wang[1], and Seung Lee[2]

[1] Division of Computer Science & Information, Daelim College
526-7, Bisan-dong, Dongan-gu, Anyang-si, Gyeonggi-do, 431-715, Korea
{pewkys,sypark,shwang}@daelim.ac.kr
[2] Dept. of Automatic System Engineering, Daelim College
526-7, Bisan-dong, Dongan-gu, Anyang-si, Gyeonggi-do, 431-715, Korea
slee@daelim.ac.kr

Abstract. Although many information hiding paradigm provides many advantages for protect important information, we try to introduce new method in concealing thing. One of these is Steganography Many efforts have been made to encrypt data as well as to hide data. Most users wish not to lose the data that they want to hide. The steganography is one of methods that users can hide data. Some steganography softwares use audio data among multimedia data. However, the commercialized audio steganography softwares have disadvantages that the existence of hidden messages can be easily recognized visually and only certain-sized data can be hidden. To solve these problems, this study suggested, designed and implemented the Dynamic Message Embedding (DME) algorithm. Also, to improve the security level of the secret message, the file encryption algorithm has been applied. Through the DME algorithm and the file encryption algorithm, StegoWaveK system that performs audio steganography was designed and implemented. Then, the suggested system and the commercialized audio steganography system were compared and analyzed on some criteria.

Keywords: Steganography, Information Hiding, CDMA, Stego-Data, Cover-Data, Wave file

1 Introduction

As computers and communication systems are rapidly developing, several methods have been introduced to keep the data safely. The basic method is encoding and the other method is data hiding techniques. The most typical application technique to hide data is the steganography and watermarking. The steganography is a technique that unauthorized users from finding out hidden data by hiding data in a variety of media that of text, image, moving image, and audio. Although an attacker might find secret messages that were encoded and hidden, generally the attacker needs to decode the message, which means higher security level of the message. Also, users tend to compress the secret data before hiding because the required data size is decided depending on the size of the secret data. Here, the data to cover the secret data is called “Cover-data”, and the data to be hidden through the steganography method is called “Stego-data”. While image-based steganography is more often used and developed than other steganography methods, audio-based steganography is also actively researched. In the audio steganography, data is generally hidden where the listener actually does not listen to. Therefore, listeners are not aware that data is actually hidden. Besides, since the Stego-data size is almost similar to the Cover-data size, most listeners cannot tell the difference.

For not only authentication and copy right but also concealing files containing the important information, many methods are appeared. We proposed the methods that hide the information on Wave files. This method extracts the information not by using original file. Thus we discriminate the file including the information and the file not including the information. Main method used on Wave files is low bit encoding, but we propose the new method using by CDMA.

CDMA(Code Division Multiple Access) is a digital spread-spectrum modulation technique used mainly with personal communications devices such as mobile phones. CDMA digitizes the conversation and tags it with a special frequency code. The data is then scattered across the frequency band in a pseudo random pattern. The receiving device is instructed to decipher only the data corresponding to a particular code to reconstruct the signal.

In order to protect the signal, the code used is pseudo-random. It appears random, but is actually deterministic, so that the receiver can reconstruct the code for synchronous detection. This pseudo-random code is also called pseudo-noise (PN). We used 8-bit PN code. We compared two methods, CDMA method and StegoWaveK, thus we know that the proposed method has similar to performance of StegoWaveK.

2 StegoWaveK

StegoWaveK model where a message is hidden using the sine accumulation function has been designed and implemented. The commercialized audio steganography software hides the secret message by one bit in the 16 bit lowbit allowing attackers to visually filter the hidden message and lowering capacity. To solve this problem, the Dynamic Message Embedding (DME) module has been designed and implemented for StegoWaveK model (the suggested audio steganography model). The DME module analyzes the number of bits hidden in every 16 bits of the cover data and the locations within Cover-data characteristics by analyzing characteristics of the Cover-data and the secret message to select the most suitable embedding function and hide the secret message.

Following Fig. 1 shows the flow of the StegoWaveK audio steganography model consisting of compression, encoding, and insertion of the secret message into the Cover-data. Users can select encoding.

In the Dynamic Message Embedding (DME) module, the secret message is inserted into a certain critical value, not the lowbit as long as characteristics of the Cover-data are maintained. For the critical value, features of the Cover-data and the secrete message are analyzed and processed. Also, the most suitable algorithm is selected to insert the secret message from the algorithm that hides one bit and the algorithm that improves capacity by the sine curve form

In the pre-processing, the sound volume of the Cover-data that is a wave file is analyzed and distribution of the sound volume of the wave file is studied. Then, referring to the ratio with the next secret message, the critical value to hide the secret message is decided. After the location to hide the secrete message is decided, the message embedding

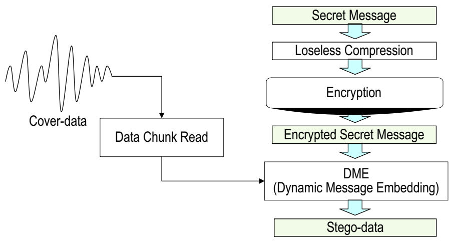


Fig. 1. Flowchart of StegoWaveK Model

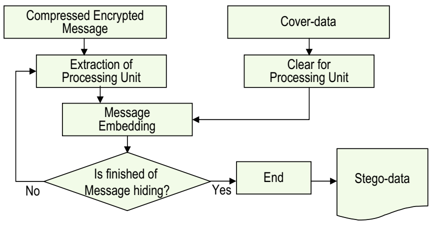

Compressed Encrypted Message
Cover-data
Extraction of Processing Unit
Clear for Processing Unit
Message Embedding
Is finished of Message hiding?
No
Yes
End
Stego-data

Fig. 2. Message Embedding FlowChart

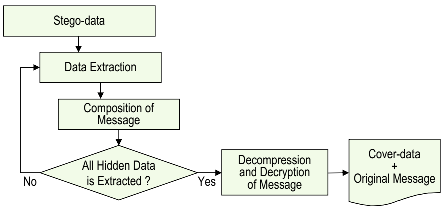

Stego-data
Data Extraction
Composition of Message
All Hidden Data is Extracted ?
No
Yes
Decompression and Decryption of Message
Cover-data + Original Message

Fig. 3. Message Extracting FlowChart

algorithm is decided. Fig. 2 and Fig. 3 show actual procedures to hide and extract the secret message.

Basically, Fmt chunk (the header of the wave file) size is 18 bytes. Last 2 bytes of these 18 bytes describe the size of extension information. However, if extension information is not included, remaining 16 bytes is designed as Fmt chunk. Actually, in most cases, Fmt chunk is designed to be 16 bytes. In the commercialized audio steganography software, only the wave file with the data chunk identifier in 4 bytes from the 37th byte (the location defined assuming that Fmt chunk of the wave file is 16 bytes) is recognized as a wave file. In other words, the commercialized audio steganography software does not recognize 18 byte wave files. Especially, when MP3 music files are converted into wave files, mostly they have 18 byte or 14 byte Fmt chunk, which is not supported by the commercialized audio steganography software. To solve this problem, the Chunk Unit Read (CUR) module processing according to the chunk of the wave file has been designed. In the CUR module, data characteristics are not judged by reading fixed location values, but instead, the wave file is processed according to the chunk. Therefore, not only the wave file with 18 byte Fmt chunk (the basic format) but also wave files with 16 byte and 14 byte Fmt chucks can be used as the Cover-data

3 CDMA Method and CDStego

This section introduces CDMA method used on information hiding, in comparison with StegoWaveK [9].

Fig. 4 shows the case where the spreading signal is demodulated by a different spreading code from the sender's. If the spreading signal is demodulated by a different spreading code having the value of '01101001010', or if it is not demodulated in time, the receiver can not decode the information data exactly. Therefore, the receiver has to know the exact spreading code and the time to recover the original information data in CDMA.

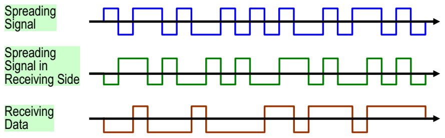


Fig. 4. The case where the spreading is demodulated by a different spreading code from the sender's

Spreading codes similar to white noises has no relevance to information data. Because spreading codes have random values and can be generated infinitely, it is next to impossible for eavesdroppers to guess the spreading codes.

Fig. 5 and Fig. 6 shows a rule and an example for generating spreading codes respectively. This rule is called as Orthogonal Variable Spreading Factor(OVSF) code.

$$\begin{array}{l}
C_{ch,1,0} = (1) \\
[C_{ch,2,0}] = [C_{ch,1,0} \quad C_{ch,1,0}] = (1,\ 1) \\
[C_{ch,2,1}] = [C_{ch,1,0} \quad -C_{ch,1,0}] = (1,-1) \\
[C_{ch,2^{(n+1)},0}] = [C_{ch,2^{(n)},0} \quad C_{ch,2^{(n)},0}] \\
[C_{ch,2^{(n+1)},1}] = [C_{ch,2^{(n)},0} \quad -C_{ch,2^{(n)},0}] \\
[C_{ch,2^{(n+1)},2}] = [C_{ch,2^{(n)},1} \quad C_{ch,2^{(n)},1}] \\
[C_{ch,2^{(n+1)},3}] = [C_{ch,2^{(n)},1} \quad -C_{ch,2^{(n)},1}],\ \text{etc.}
\end{array}$$

Fig. 5. Rule for generating spreading codes

Let us look at CDMA more closely. A CDMA encoder $Z_{i,m}$ can be represented by the multiplication of the ith data bit d_i and the mth CDMA code c_m as shown below. After receiving the encoded bit of $Z_{i,m}$ without interferences, a receiver can decode the original data bit d_i through the equation as shown below.

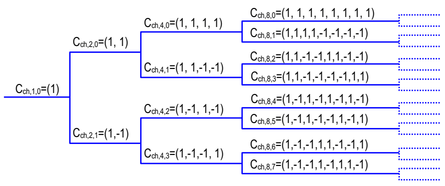

Cch,1,0=(1)
Cch,2,0=(1, 1)
Cch,2,1=(1,-1)
Cch,4,0=(1, 1, 1, 1)
Cch,4,1=(1, 1,-1,-1)
Cch,4,2=(1,-1, 1,-1)
Cch,4,3=(1,-1,-1, 1)
Cch,8,0=(1, 1, 1, 1, 1, 1, 1, 1)
Cch,8,1=(1,1,1,1,-1,-1,-1,-1)
Cch,8,2=(1,1,-1,-1,1,1,-1,-1)
Cch,8,3=(1,1,-1,-1,-1,-1,1,1)
Cch,8,4=(1,-1,1,-1,1,-1,1,-1)
Cch,8,5=(1,-1,1,-1,-1,1,-1,1)
Cch,8,6=(1,-1,-1,1,1,-1,-1,1)
Cch,8,7=(1,-1,-1,1,-1,1,1,-1)

Fig. 6. Example for generating spreading codes

When N senders transmit data simultaneously without interferences, A Mixed CDMA encoder $Z^*_{i,m}$ can be represented as shown below. As the same case of the single sender's example, receivers can decode the original data bit d_i through the equation.

The interfaces of the encoding process consist of five steps including the additional step 4, but the interface of the decoding process has not been changed in comparison with the previous case.

The interface for the step 4 has a checkbox option which describes whether the CDMA encoding function has been selected or not. It also has radio buttons which describe the size of a Pseudo-random Number(PN) code. There are three types of PN code like 8 bits, 16 bits, and 32 bits. We will now examine the encoding process more closely. When the size of data is 8 bits, 8 bits PN code can be generated as follows by using the generation rule.

When the 8 bit data have the value of '10110010', if the value of '0' is represented as '-1', the data can be expressed as (1,-1,1,1,-1,-1,1,-1). Then, the data can be encoded by the 8 bit PN code as shown above. Detailed encoding process is shown in Fig. 7.

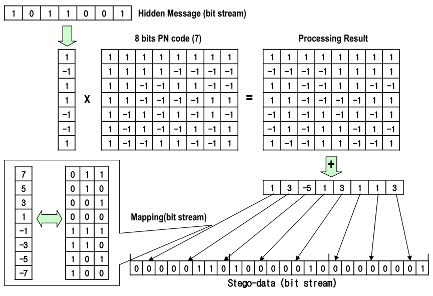


Fig. 7. CDMA encoding process when the number of the PN codes is 7

If we sum up the above result along the column, the calculation can be expressed as (0, 4,-4, 0, 4, 0, 0, 4), and then it would be stored to the memory as a bit stream. The calculation consists of eight numbers, and each number of the calculation ranges from -4 to 4. But, If all the same numbers in the column are 1 or -1 respectively, each number of the calculation can range from -8 to 8. Each number actually is one of the nine values as (-8, -6, -4, -2, 0, 2, 4, 6, 8), because the PN code consists of symmetrical 1 or -1 as shown above. Since the size of the required memory is 4 bits to express the nine values, the total size of the required memory is 32 bits (8 bits $\times$ 4 bits) to encode 8 bits data by the 8 bits PN code.

Four bits memory can include fifteen kinds of numbers. However, since the memory for the calculation only includes nine kinds of numbers, the memory for the other seven kinds of numbers becomes useless. To improve efficiency of the memory, if the last code of the above PN codes does not be used, each number of the calculation is one of the eight values as (-7, -5, -3, -1, 1, 3, 5, 7). Since the size of the required memory is 3 bits to express the eight values, the total size of the required memory is 24 bits (8 bits $\times$ 3 bits). And, if we apply only seven PN codes to the above example, the calculation can be expressed as (1, 3,-5, 1, 3, 1, 1, 3).

For example, when the number of the PN codes is 7, the total size of the required memory is 48 bits (8 bits $\times$ 3 bits $\times$ 2 times) to encode 8 bits data because the remaining 1 bit is encoded at the second time. In the case of processing 8 bits data, the total size of the memory encoded by the seven PN codes is bigger than the total size of the memory encoded by the eight PN codes.

However, in the case of processing 56 bits data, firstly, when the number of the PN codes is 8, the total size of the required memory is 224 bits (8 bits $\times$ 4 bits $\times$ 7 times). Secondly, when the number of the PN codes is 7, the total size of the required memory is 192 bits (8 bits $\times$ 3 bits $\times$ 8 times). Therefore, the total size of the memory encoded by the seven PN codes is smaller than the total size of the memory encoded by the eight PN codes.

In the decoding process, the PN codes can be generated in the same way as shown in Fig. 5 and Fig. 6. Let us decode the fifth bit from the received bit stream shown in Fig. 7.

The bit stream is divided into 3 bits fragments as shown in Fig. 8. Each fragment is mapped with a value in the mapping table. After mapping, the received bit stream is represented as (1,3,-5,1,3,1,1,3). As shown below, we multiply the mapped values by the fifth PN code expressed as (1,-1,1,-1,1,-1,1,-1) along the column. And then, we can get the numbers as (1,-3,-5,-1,3,-1,1,-3).

Let us sum up the numbers as (1,-3,-5,-1,3,-1,1,-3). The sum of the numbers is -8 as shown below.

The sum having the value of -8 is divided by 8 because the size of the PN code is 8. And we can get the fifth original bit, 0 corresponding to -1. Fig. 8 shows the CDMA decoding process when the number of the PN codes is 7.

We define required classes and functions for CDMA as shown below.

- CBitData class : This class provides functions for the conversion between bit and byte to process the modulation between an original data and a spreading bit stream. This class is similar to the existing CBitByteData class.
- EncodeSize and DecodeSize functions : These functions provides operations for calculating the size of the result data in the encoding/decoding process.
- EncodeCDMA, DecodeCDMA functions : These functions provides operations for encoding to CDMA or decoding from CDMA.

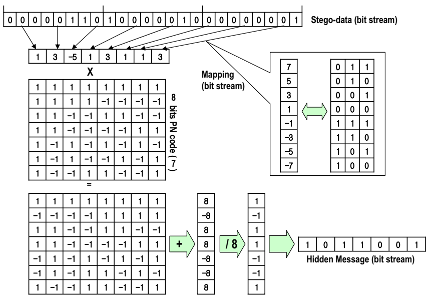


Fig. 8. CDMA decoding process when the number of the PN codes is 7

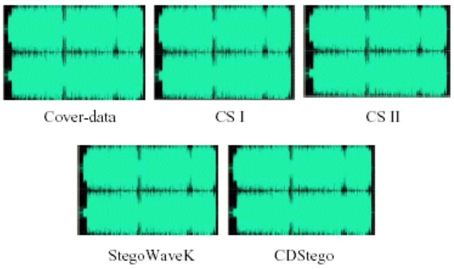


Fig. 9. Waveform Comparison of Cover-data and applied Audio Steganography WAVE Files

4 Performance Evaluation of CDStego

In this section, StegoWaveK that has been implemented by VC++. Net is compared with Invisible Secretes 2002 (hereinafter to be referred to as “CS I”) and Steganos Security Suite 4 (hereinafter to be referred to as “CS II”) that have been commercialized and in use now. According to [7], in steganography analysis, visual, audible, structural, and statistical techniques are used. Therefore, the comparison and the analysis in this study were based on criteria of the Human Visible System (HVS), Human Auditory System (HAS), Statistical Analysis (SA), and Audio Measurement (AM). Since the HAS can be relatively subjective, audio measurement analysis was added to more objectively analyze and compare the Stego-data and the Cover-data. For comparison and analysis data, the song of Yun Do Hyen’s Dolgo was used. In experiments with other genre music, similar results were gained.

According to the waveform analysis result of the Stego-data created by Invisible Secretes 2002 CDStego, and StegoWaveK using an audio editor, CoolEdit 2000, it is hard to visually tell the difference due to HVS characteristics. Fig. 9 shows waveform of the Stego-data captured by CoolEdit 2000.

As showed in Fig. 9, it is extremely difficult to tell whether data is hidden or not only by visually checking waveform.

To analyze and compare the suggested system with the existing system on criteria of the Human Auditory System (HAS), 13 messages with different sizes and 4 wave files

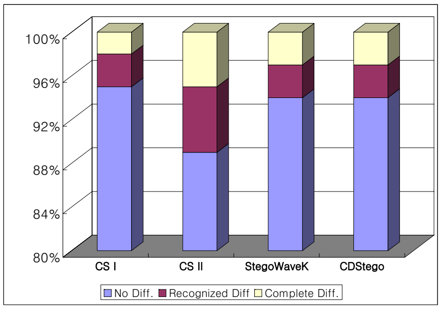


Fig. 10. Result of WAVE Listening Test

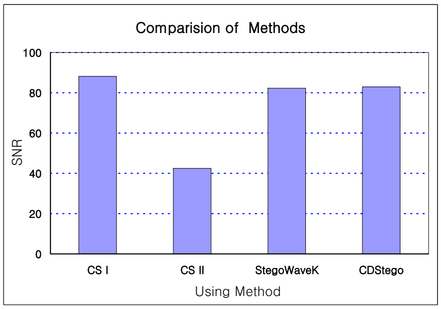


Fig. 11. Comparison of SNR(dB) between Cover-data and Stego-data

were selected as Cover-data. We played the Stego-data where the message is hidden through CS I and CS II using lowbit encoding and the Stego-data where the message is hidden through StegoWaveK system and CDStego to 100 students. Although it could be subjective, most students could not tell the difference between the Cover-data and the Stego-data. Following Fig. 10 shows the experiment result.

Audio measurement and analysis includes frequency response, gain or loss, harmonic distortion, intermodulation distortion, noise level, phase response, and transient response. These parameters include the signal level or phase and the frequency. For example, the Signal to Noise Ratio (SNR) is a level measurement method represented by dB or ratio. In [4, 5], the quality of the Stego-data where the message is hidden was measured using SNR. SNR represents ratios of relative values [1, 2].

The following graph shows SNRs between the cover data and the Stego-data created by the CS I, the CS II, CDStego, and StegoWaveK. The SNR of the Stego-data created by the suggested system is not significantly different from that of the Stego-data created by the CS I. However, the SNR of the Stego-data created by the CS II is relatively different from that of the Stego-data created by the suggested system.

In the following, the Cover-data and the Stego-data have been analyzed by the Perceptual Evaluation of Speech Quality (PESQ). It is difficult to completely trust the result of the automated sound measurement such as PESQ, but the result has enough reliability to be used in various kinds of related tests [6].

5 Conclusion

The audio steganography that hides data uses the wave file as Cover-data; the Cover-data has the same size as the size of the Stego-data file in the suggested system; and most listeners and users do not tell any difference in the sound quality. Therefore, they cannot recognize that data is hidden in Stego-data. Also, none of the HVS system or the HAS system can analyze the wavelength that had been analyzed through a simple audio editing tool in an intuitive way. Therefore, it can be useful to send secret data.

The commercialized steganography software uses only certain types of wave files as the Cover-data. As one bit of the secret message is inserted into the Lowbit of the Cover-data, the cover data can be easily filtered. Also, as one bit is inserted to hide the secrete message, the Cover-data size increases.

CDStego model suggested in this study has been specially designed to conceal data more safely, and improve problems of existing commercial audio steganography softwares. Thus CDStego has similar property to StegoWaveK [9] and more large size of hidden message.

While data is only visually hidden through characteristics of the file in Windows O/S, CDStego model hides data in the audio data that is often used. Although the hidden data does not exist in the computer any longer, the user can extract data whenever he/she wants without any loss of the hidden data. Therefore, CDStego and StegoWaveK model can be useful to hide important design drawings, program files, and confidential documents.

More studies shall be performed relating to migration to the embedded system in the future. By introducing new methods suggested in [8], or by using loss-free compression programs such as TIFF with high performance, it would be possible to improve performance of CDStego.

References

1. S.K. Pal, P.K. Saxena, S.K. Muttoo, "The Future of Audio Steganography", *STEG'02*, July 11-12, 2002.
2. http://www-ccrma.stanford.edu/CCRMA/Courses/422/projects/WaveFormat
3. Fabien A.P. Petitcolas, Ross J. Anderson, and Markys G.Kuhn, "Information Hiding - A Survey", *Proceedings of the IEEE, special issue on protection of multimedia content*, 87(7):1062-1078, July 1999.
4. Stefan Katzenbeisser and Fabien A.P.Petitcolas "Information hiding techniques for steganography and digital watermarking", *Artech House Publishers*, 2000.
5. J.D.Gordy and L.T.Bruton, "Performance Evaluation of Digital Audio Watermarking Algorithms.", *IEEE MWSCAS 2000.*
6. Djimitri Wiggert, "Codes for Error Control and Synchronization", *Artech House Inc*, 1988.
7. Peter Wayner, "Disappearing cryptography Information Hiding : Steganography & Watermarking", *second edition, chapter 17, Morgan Kauffman*, 2002.
8. Ira S. Moskowitz, Garth E. Longdon, and LiWu Chang, "A New p aradigm Hidden Steganography", *New Security Paradigms workshop 2000*, September, 19th 21st, 2000, Cork Ireland.
9. Y.S. Kim at el, "Information Hiding System StrgoWaveK for Improving Capacity", *The 2003 Internationbal Symposium on Parallel and Distributed Processing and Applications.*

Efficient Key Distribution Protocol for Electronic Commerce in Mobile Communications*

Jin Kwak[1], Soohyun Oh[2], and Dongho Won[1]

[1] Information and Communications Security Lab.
School of Information and Communication Engineering
Sunkyunkwan University
300 Choencheon-Dong, Jangan-Gu, Suwon, Gyeonggi-Do, 440-746, Korea
{jkwak,dhwon}@dosan.skku.ac.kr
[2] Division of Computer Science
Hoseo University, Asan, Chuncheongnam-Do, 336-795, Korea
shoh@office.hoseo.ac.kr

Abstract. Recently, the development of mobile communication technology, the mobile terminal users are increasing. In the mobile environment, the security service is very important to provide secure mobile communication, such as mobile electronic-commerce. The security of information is significant to provide secure electronic commerce in mobile communications. Therefore, the security services must be provided in a mobile communication environment. However, the mobile communication environment has some disadvantages, because it has low capacity of power, low performance of CPU and limited memory, etc. In this paper, we propose an efficient key distribution protocol for mobile communications.

1 Introduction

As the development of mobile communication technology, the mobile terminal users are increasing. In result, Internet services and applications for mobile terminals are increasing rapidly. In order to provide these services and applications over the mobile communications, the security services are very important to provide secure mobile communication. Therefore, the security services such as confidentiality, authentication, integrity, and non-repudiation are must be provided in mobile communication environment same as the wired Internet.

However, mobile communication environment has some disadvantages, because it has low capacity of power, low performance of CPU and limited memory, etc. The security technology for secure mobile communication is applicable to the ubiquitous computing domain in the near future. The one of the famous protocols are provided [1,2,3,4,6,7,10,11,12,13], however proposed previous protocols has some weaknesses for active attacks.

Therefore, in this paper, we consider the active attacks, and then propose the efficient key distribution protocol in mobile environment which is secure against active attacks.

* This work was supported by the University IT Research Center Project by the MIC(Ministry of Information and Communication) Republic of Korea.

The goal of this paper is introduce the simple and efficient key distribution method, but it secure against active attacks. Then we propose an efficient key distribution protocol for electronic commerce in mobile communications(M-Commerce).

This paper is organized as follows, first we will describe the security requirements of the mobile communication environment, and then we propose efficient key distribution protocol for M-Commerce. Next, we analyze the securities and properties of the proposed protocol comparison with existing protocols. Finally, we will make a conclusion.

2 Security Requirements Primer

For secure communication in the mobile environment, the mobile key distribution protocol have to satisfied with the below requirements [6].

- *n-pass*: The protocol that needs n number of passes to compute the session key from one party's point of view. The less n-pass is providing efficiency.
- *mutual entity authentication*: One party is assured, through the acquisition of corroborative evidence, the identity of a second party involved in a protocol, and that the second party has actually participated in the key agreement protocol.
- *secret key agreement*: To protect transaction data between the client and the server, secret key generated by the information both entities.
- *key authentication*: One party is assured that no other party aside from a specifically identified second party could possibly compute the shared secret key.
- *key freshness*: A key used to compute a shared secret value is updated after the execution of a new exchange protocol.
- *user anonymity*: It is the state of being unidentifiable within a set of subjects. It is the confidentiality of the user identity within the communication.
- *non-repudiation*: It is prevents an entity from denying previous actions, such as send or received data. When dispute arise due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. It is a necessary security requirement in electronic commerce application.

3 Proposal Protocol

In this section, we propose key distribution method for secure M-Commerce. It is secure against some active attacks(see details 4.1), also it can provide efficiencies(see details 4.2). The proposal protocol consists of the registration and key distribution phases, and the notations are shown below.

[*Notations and Definitions*]

- i : it is an entity, the server or the client
- p : a prime defining the Galois field $GF(p)$
- q : a prime factor of $p-1$
- g : a primitive element of the q-order in Z_p
- P_i : entity i's public key, $P_i = g^{S_i} \bmod p$

- S_i : entity i's secret key
- r_i : random number selected by entity i
- SIG_i : signature of the entity i
- ID_i : identification information of the entity i
- meta-ID_C : temporary ID of the client
- PIN_C : personal identification number of the client
- $h()$: hash function
- $A \parallel B$: concatenation A and B

[*Registrations*]

When the client starts communication using the mobile device, he needs registration process to the server(e.g. the shopping mall or mobile station). Fig.1. shows the process of the registration in the proposed protocol.

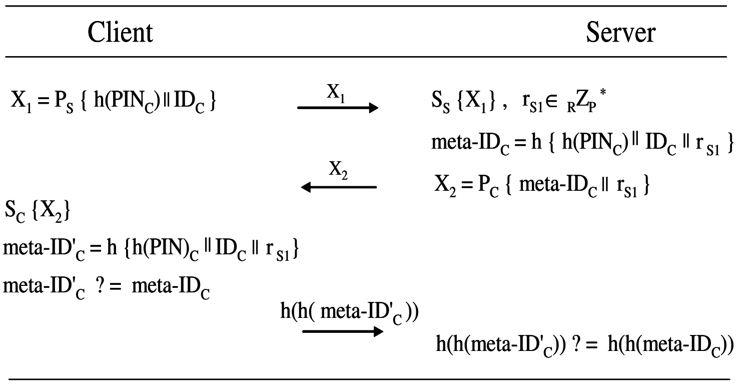

Client
Server
$X_1 = P_S\{h(PIN_C) \parallel ID_C\}$
X_1
$S_S\{X_1\}$, $r_{S1} \in_R Z_P^*$
meta-$ID_C = h\{h(PIN_C) \parallel ID_C \parallel r_{S1}\}$
X_2
$X_2 = P_C\{$meta-$ID_C \parallel r_{S1}\}$
$S_C\{X_2\}$
meta-$ID'_C = h\{h(PIN)_C \parallel ID_C \parallel r_{S1}\}$
meta-ID'_C ?= meta-ID_C
$h(h($meta-$ID'_C))$
$h(h($meta-$ID'_C))$?= $h(h($meta-$ID_C))$

Fig. 1. Registration phase

1. The client selects PIN_C, and computes X_1 using own ID_C and $h(PIN_C)$.

$$X_1 = P_s\{h(PIN_c) \parallel ID_c\}$$

2. The server decrypts received X_1 using his secret key S_S, and select random number r_{S1}. The server computes temporary meta-ID_C and X_2 after stores r_{S1} and $h(PIN_C)$, then he sends X_2 to the client.

$$X_2 = P_c\{meta - ID_c \parallel r_{s1}\}$$

3. When the client received X_2, he decrypts it using his secret key S_C. Then he computes temporary meta-ID'_C using the server's random number r_{S1}. If it equals to received meta-ID_C, the client sends hashed value to the server.

$$meta - ID'_c \overset{?}{=} meta - ID_c$$

4. The server checks hashed meta-ID'_C equals to hashed meta-ID_C generated by the server.
$$h(h(meta - ID'_c)) \overset{?}{=} h(h(meta - ID_c))$$

[*Key Distribution*]

In the previous protocols, the client authenticates the server, but the server cannot authenticate the client(one-way authentication). Therefore, the mutual entity authentication is needed to secure mobile communication [8]. The operation of the propose protocol is as follows. Fig.2 shows the key distribution process of the propose protocol for secure mobile communication.

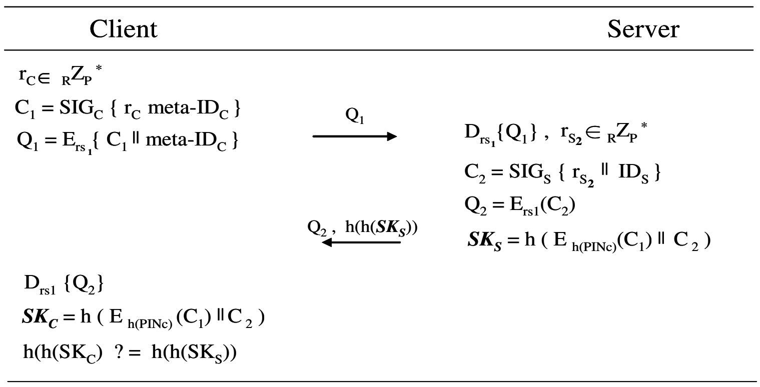

Client
Server
$r_C \in_R Z_P^*$
$C_1 = SIG_C\{ r_C$ meta-$ID_C \}$
$Q_1 = E_{rs_1}\{ C_1 \| $ meta-$ID_C \}$
Q_1
$D_{rs_1}\{Q_1\}$, $r_{S_2} \in_R Z_P^*$
$C_2 = SIG_S\{ r_{S_2} \| ID_S \}$
$Q_2 = E_{rs1}(C_2)$
$SK_S = h(E_{h(PINc)}(C_1) \| C_2)$
Q_2, $h(h(SK_S))$
$D_{rs1}\{Q_2\}$
$SK_C = h(E_{h(PINc)}(C_1) \| C_2)$
$h(h(SK_C)) ?= h(h(SK_S))$

Fig. 2. Key Distribution phase

1. The client selects random number r_C, and computes C_1 with his signature. Then he computes Q_1 and sends it to the server.
$$Q_1 = E_{r_{s1}}\{C_1 \parallel meta - ID_c\}$$

2. The server decrypts received Q_1 and select random number r_{S2}. The server computes C_2, Q_2, and generates session key SK_S. Then he sends Q_2 and hashed value of session key SK_S.
$$Q_2 = E_{r_{s1}}(C_2)$$
$$SK_s = h\{E_{h(PIN_c)}(C_1) \parallel C_2\}$$

3. The client decrypts Q_2, and computes session key SK_C. Then he checks the received hashed SK_S equals to hashed SK_C computed by the server.
$$SK_c = h\{E_{h(PIN_c)}(C_1) \parallel C_2\}$$
$$h(h(SK_c)) \overset{?}{=} h(h(SK_s))$$

4 Analysis

In this section, we analyze the securities and properties of the proposed protocol. Proposed protocol is based on the difficulty of the discrete logarithm problem, therefore an attacker computes the session key using public information, which is the equivalent to solve the difficulty of the discrete logarithm based problem. For analysis of the securities, we considered some active attack models [9].

[*Securities*]

Definition 1. Active Impersonation:
An adversary enters a session with a legitimate entity, impersonates another entity, and finally computes a session key between entities.

Any adversary who does not know the secret information of each entity, and masquerades as an other entity, participates in the protocol and successfully shares a session key with the valid entity. The proposed protocol uses the hash value and secret information, therefore, an attack is impossible since the adversary should not know the secret key of each entity, and randomly selected r_{S2} to compute the valid session key.

Definition 2. Forward Secrecy:
The secrecy of the previous session keys established by a legitimate entity is compromised. A distinction is sometimes made between the scenario in which a single entity's secret key is compromised(half forward secrecy), and the scenario in which the secret key of both entities are compromised(full forward secrecy).

If the secret key of the client is exposed and the adversary obtains this, the adversary cannot compute the previous session key. The adversary should obtain the hash value $h(PIN_C)$ and randomly selected r_{S2} in order to compute the session key, therefore the attack is impossible. In contrast, if the secret key of the server is exposed, the adversary cannot compute the previous session key. The adversary should obtain the hash value $h(PIN_C)$ and randomly selected r_{S2} in order to compute the session key, therefore the attack is impossible.

The attacker should get secret key of each entity, in order to compute the previous session key, and since it is equivalent to difficulty of the passive attacker because an adversary does not know the hash value $h(PIN_C)$ and randomly selected r_{S2}, therefore the attack is impossible. Accordingly, the proposed protocol satisfies the *full forward secrecy* since the session key is secure even if the old secret key of each entity is exposed.

Definition 3. Key-Compromise Impersonation:
Suppose the secret key of an entity is disclosed. Clearly an adversary that knows this value can now impersonate an entity, since it is precisely this value that identifies the entity. In some circumstances, it maybe desirable that this loss does not enable the adversary to impersonate the other entity, and share a key with the entity as the legitimate user.

In the proposed protocol, an adversary cannot masquerade as a client to the server or viceversa, even if the secret key of the client is exposed because an adversary cannot know the $h(PIN_C)$ and r_{S2}. Therefore, an adversary cannot masquerade as a valid client even if he obtains the secret key.

Definition 4. Known Key Secrecy:
A protocol should still achieve its goal in the face of an adversary who learns some other session keys. A protocol is said to provide a known key security if this condition is satisfied.

Since the proposed protocol uses a random value r_{S2} chosen by the server and r_C chosen by the client for each session in order to compute a session key, it cannot give any advantage in obtaining the session key of the current session, even if the previous transmission information are exposed. Therefore, the difficulty of a known key passive attack is the same as the passive attacker who has no information on a previous session. It is also impossible for a known key impersonation attack, because an adversary cannot go directly or participate in the session. The adversary cannot masquerade as the server or the client using the transformation information of the current session, the transformation information of the previous session, or to set the session key.

[*Properties*]

We analyze the efficiency of the proposed protocol, and compare it with the existing protocol. Concerning the efficiency analysis of the proposed protocol, we will consider the properties security requirements(see details 2.) [5,9]. Table 1. shows the characteristics of the proposed protocol in comparison with the existing protocol.

The proposed protocol is a 2-pass protocol same as previous protocol. The previous BCY [1] does not provide an entity authentication, and PACS [10] provides a one-way entity authentication. On the other hand, the proposed protocol provides mutual entity authentication since the server and the client generates session key using secret information belongs to the client and the server.

When the computes session key, proposed protocol provide implicit key authentication to the client, and provide explicit key authentication to the server. It is same as PACS protocol, but BCY provides explicit key authentication to each other.

The proposed protocol provides mutual key freshness, since they compute session key with random values chosen by each other. In addition, BCY and PACS provides one-way user anonymity, but proposed protocol provides mutual anonymity. The proposed protocol also provides mutual non-repudiation service since using other party's signatures when they compute the session key.

5 Conclusion

In this paper, we proposed secure and efficient key distribution protocol in mobile environment, which is secure against active attacks such as active impersonation, forward secrecy, key-compromise impersonation, and known key secrecy. Furthermore, the proposed protocol can provide mutual entity authentication, key authentication, mutual key freshness, user anonymity, and non-repudiation. As a result, the proposed protocol will

Table 1. Properties of proposed protocol in comparison with previous protocol

properties	BCY	PACS	proposed protodcol
n-pass	2	2	2
entity authentication	×	one-way	mutual
key authentication	user : explicit	user : implicit	user : implicit
	server : explicit	server : explicit	server : explicit
key freshness	×	×	mutual
user anonymity	one-way	one-way	mutual
non-repudiation	×	one-way	mutual

be useful to electronic commerce using mobile terminals. Furthermore, the proposed protocol will be useful to ubiquitous computing environments, because the proposed protocol is simple but secure. The proposed protocol can be implemented efficiently, and provides the additional properties of security and efficiency which have been scrutinized in this paper.

References

1. M.J. Beller, L.F. Chang, and Y. Yacobi. Security for Personal Communication Service: Public-key vs. Private key approaches. *Proceedings of Third IEEE International Symposium on Personal, Indoor and Mobile Radio Communications–PIMRC'92*, pages 26–31, 1992.
2. M.J. Beller, L.F. Chang, and Y. Yacobi. Privacy and authentication on a portable communications system. *IEEE Journal on Selected Areas in Communications*, 11(6):821–829, 1993.
3. M.J. Beller and Y. Yacobi. Fully-fledged two-way public key authentication and key agreement for low-cost terminals. *IEE Electronics Letters*, 29(11):999–1001, 1993.
4. U. Carlsen. Optimal Privacy and Authentication on a Portable Communications System. *ACM Operating Systems Review*, 28(3):16–23, 1994.
5. J. S. Go and K. J. Kim. Wireless authentication Protocols Preserving User Anonymity. *Proceedings of SCIS 2001*, 2(1):159–164, 2001.
6. G. Horn, K. M. Martin and C. J. Mitchell. Authentication protocols for mobile network environment value-added services. *IEEE Transactions on Vehicular Technology*, 51:383–392, 2002.
7. K. H. Lee and S. J. Moon. AKA Protocols for Mobile Communications. *Australasian Conference in Information Security and Privacy–ACISP 2000, number 1841 in Lecture Note in Computer Science Number*, pages 400–411, 2000.
8. K. Nyberg and R. A. Rueppel. Message recovery for signature scheme based on the disscrete logarithm problem. *Advacne in Cryptography–EUROCRYPT'94, number 950 in Lecture Note in Computer Science Number*, pages 182–193, 1995.
9. S. H. Oh, J. Kwak, S. W. Lee, and D. H. Won. Security Analysis and Applications of Standard key Agreement Protocols. *Computational Science and Its Applications–ICCSA 2003, number 2668 in Lecture Note in Computer Science Number, part 2*, pages 191–200, 2003.
10. PACS. PACS(Personal Access Communications System) Air Interface Standard. *J-STD-014*, 1995.

11. V. Varadharajan, and Y. Mu. On the Design of Security Protocols for Mobile Communications. *Australasian Conference in Information Security and Privacy–ACISP'96*, pages 134–145, 1996.
12. Y. Zheng. An Authentication and Security Protocol for Mobile Computing. *Proceedings of IFIP*, pages 249–257, 1996.
13. Y. Zheng. Digital Signcryption or How to Achive Cost(Signature and Encryption) $<<$ Cost(Signature) $+$ Cost(Encryption). *Advacne in Cryptography–CRYPTO'97, number 1294 in Lecture Note in Computer Science Number*, pages 165–179, 1997.

A Framework for Modeling Organization Structure in Role Engineering*

HyungHyo Lee[1], YoungLok Lee[2], and BongNam Noh[2]

[1] Div. of Info. and Electronic Commerce Wonkwang Univ., Iksan, 570-749, Korea
hlee@wonkwang.ac.kr

[2] Dept. of Computer Science Chonnam National Univ., Gwangju, 500-757, Korea
{yrlee,bongnam}@chonnam.ac.kr

Abstract. RBAC model is renowned as a security model for corporate environment, since its components, especially role hierarchy, are suitable for modeling an organization structure. But the functional role hierarchy constructed through the existing role engineering approaches does not reflect an organization structure, because they do not take the structural characteristics of the organization into account. Also, it has been observed that the unconditional permission inheritance property in functional role hierarchy may breach a least privilege security principle and make it impossible to define separation of duty requirements on roles that have a common senior role. In this paper, we propose a role engineering methodology considering organizational roles as well as functional roles to provide a practical RBAC model for corporate environment. We also elaborate the characteristics of organizational roles relatively neglected in the previous work, and compare them with those of functional roles. And models for associating organizational and functional roles and those role hierarchies (unified vs. separate) are proposed and the advantages and shortcomings of those models are given.

1 Introduction

Role-based Access Control (RBAC) is now widely accepted as an alternative to the traditional access control models, such as MAC and DAC. The basic idea of associating a set of privileges or permissions with a named role and assigning that role to users is well established and is deployed several commercial computer systems and applications [6].

While the concepts of a *role* and a *role hierarchy* are central to many RBAC models, most of the research on RBAC has focused on the representation, administration and activation of roles, assuming that roles and role hierarchy are already determined [8]. Recently, a number of researches have been conducted to develop a methodology for extracting the main components of RBAC model such as roles and role hierarchy from organization's business processes and documents [3, 4, 7, 8]. *Role engineering* for RBAC is the process of defining roles, permissions, role hierarchies, constraints and assigning the permissions to the roles [9]. It is the first step to implement RBAC system and also a crucial component of security engineering. After conducting researches on role engineering, researchers experienced that there exist different role classes such as *functional,*

* This paper was supported by Wonkwang University in 2004.

organizational, hierarchical etc., and suggested that those roles are needed for modeling a given organization's security policy completely. But, little research clearly explains how those hierarchies are interrelated and proposes a role engineering methodology employing more than one role class. Most of the researches dealt with the processes for deriving only functional roles.

RBAC model is renowned as a security model for corporate environment, since its components, especially role hierarchy, are suitable for modeling an organization structure. But the functional role hierarchy constructed through the existing role engineering approaches does not reflect an organization structure, because they do not take the structural characteristics of the organization into account. Also, it has been observed that the unconditional permission inheritance property in functional role hierarchy may breach a least privilege security principle and make it impossible to define separation of duty requirements on roles that have a common senior role. Researchers found that elements of an organization structure such as departments and teams inherently provide a important means for deriving functions (i.e., permissions) and a natural interface for managing user-to-role assignment task for security administrator. For example, departments *Personnel* and *Accounting* have their own functionalities, and organizational roles *Personnel Manager* and *Accounting Director* are authorized in the context of their departments' tasks. In addition, it is natural for security administrator to assign a user to organizational role, i.e., *Personnel Manager* rather than to several functional roles, such as *Personnel Information Management* and *Personal Rating Information Management*, which are the job functions of the *Personnel Manager* role. Unlike functional role hierarchy, a senior role in organizational role hierarchy need not inherit the permissions of its junior roles, since the inheritance scope of organizational role hierarchy is typically bound to their structural elements.

In this paper, we propose a role engineering methodology considering organizational roles as well as functional roles to provide a practical RBAC model for corporate environment. We also elaborate the characteristics of organizational roles relatively neglected in the previous work, and compare them with those of functional roles. And models for associating organizational and functional roles and those role hierarchies (*unified* vs. *separate*) are proposed and the advantages and shortcomings of those models are given. The rest of this paper is organized as follows: Section 2 describes the related work. Section 3 describes the methodology of modeling RBAC with organization structure from two viewpoints. And finally this paper concludes with analyzing the framework suggested with conventional framework and describing the further work.

2 Related Works

In [6], the RBAC framework is extended to include the notion of role hierarchies. The model allows the occupants of superior roles to inherit all the positive access rights of their inferiors, and conversely ensures that the occupants of inferior positions inherit any prohibitions that apply to their superiors. However, the authors observed that in some situations inheritance of access rights down the organizational hierarchy may be undesirable. They outline two possible ways of avoiding this. First, they use some other ordering than the organizational hierarchy to define the role hierarchy. Second, they define subsidiary ("*private*") roles outside the hierarchy.

In [7], the authors used a generalization (*is-a* relationship) hierarchy based on professional competencies for the analysis of role hierarchies. On the other hand [8] used a generalization hierarchy based on an organization's functional hierarchy. In [4], they outlined the control principles which are applied in many large organizations and their impact on inherited access rights, and came to the conclusion that the interaction of control principles and role hierarchies could have undesirable consequences for access control. They pointed out that the generalization hierarchy is not the only one that could be validly used for a role hierarchy. That paper was limited mainly to pointing out the problems.

In [4], they describe the work in progress with a process-oriented approach for role-finding to implement Role-Based Security Administration. They pointed out that there are a few authorizations which do not belong to functional roles. To overcome that problem, they have agreed on a few nonfunctional role-classes to be introduced to the RBAC concept. The roles are organized within different role classes. Role classes defined at Siemens ICN are *functional, organizational, basic, hierarchical, and special* roles.

3 A Framework for Modeling Organization Structure

As earlier researches show, functional roles alone can not model the authorization characteristics of an organization and does not provide an intuitive mapping between an organizational structure and role hierarchies.

In this section, we propose two different approaches to solve those problems. The first one is a *unified role hierarchy* approach. In this methodology, functional sub-role hierarchies and organizational sub-role hierarchy are interconnected and formed into a unified role hierarchy. The other approach is based on *separate role hierarchy*. In the methodology, rather than combining functional and organizational roles, two separate role hierarchies are modeled and managed: the functional role hierarchy is constructed in terms of permission inheritance for efficient permission management while the organization role hierarchy is modeled to reflect the organization structure.

3.1 Unified Role Hierarchy Approach

We briefly describe the notion of sub-role and sub-role hierarchy, key characteristics of the unified role hierarchy approach. They provide the functional (i.e., *job function*) and organizational (i.e., *job position*) authorization features through inter-related sub-role hierarchies. The detailed components and features of the model are in [5].

3.1.1 Roles and Sub-roles. A role is a job function in the organization that describes the authority and responsibility conferred on a user assigned to the role [6]. Administrators can create roles according to the job functions performed in their organizations. This is very intuitive and efficient approach except unconditional permission inheritance. In RBAC models, a senior role inherits the permissions of all its junior roles. This property may breach the least privilege principle, one of the main security principles RBAC models support [3].

In order to address this drawback, we divide a role into a number of *sub-roles* based on the characteristics of *job functions* and *the degree of inheritance*. Figure 1 shows a proposed sub-role concept compared with an original or *macro* role.

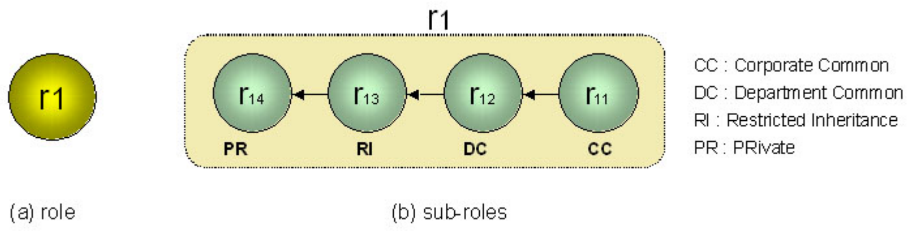


Fig. 1. Sub-role concept for corporate environment

3.1.2 Permission Inheritance in Sub-role Hierarchies. There are two kinds of sub-role hierarchies: *horizontal* and *vertical*. Horizontal sub-role hierarchy (i.e., *job position plane*) is a partially ordered relation between sub-roles which are divided from the same macro role, and only unconditional permission inheritance exists between sub-roles. Vertical sub-role hierarchy (i.e., *job function plane*) is a partially ordered relation between sub-roles in the same sub-role category (i.e., CC, DC, RI, PR sub-roles) but in different macro roles, and both unconditional and restricted inheritance exit in it. Sub-roles in RI category need to specify the sub-roles to which their permissions should be inherited. Only the specified sub-roles in RI hierarchy can inherit the permissions of their junior sub-roles.

In Figure 2, even if sub-role r_{i3} belongs to *General Manager role*, a senior role to *Manager* role, the permissions of r_{j3} are not inherited to r_{i3}.

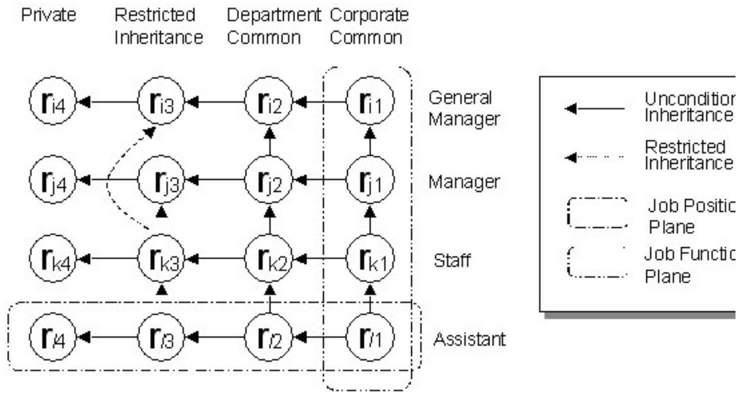


Fig. 2. Example of sub-role hierarchies

Figure 3 shows an example of an organization's structure and its corresponding sub-role hierarchies. A sub-role hierarchy in the same horizontal plane represents a job position role in the organization unit such as the *Project Leader* of Project Team 1. And, a sub-role hierarchy in the same vertical plane such as CC, DC, PI, PR is analyzed and constructed from the perspective of permission inheritance. As we can see, vertical planes in (b) correspond to organization units in (a). Users are assigned to private sub-roles since the private sub-role inherits all permission of its junior roles. Despite of its complex structure of sub-role hierarchies, sub-role hierarchy approach provides a role engineering approach for analyzing functional and organizational roles and their relationships to security administrators.

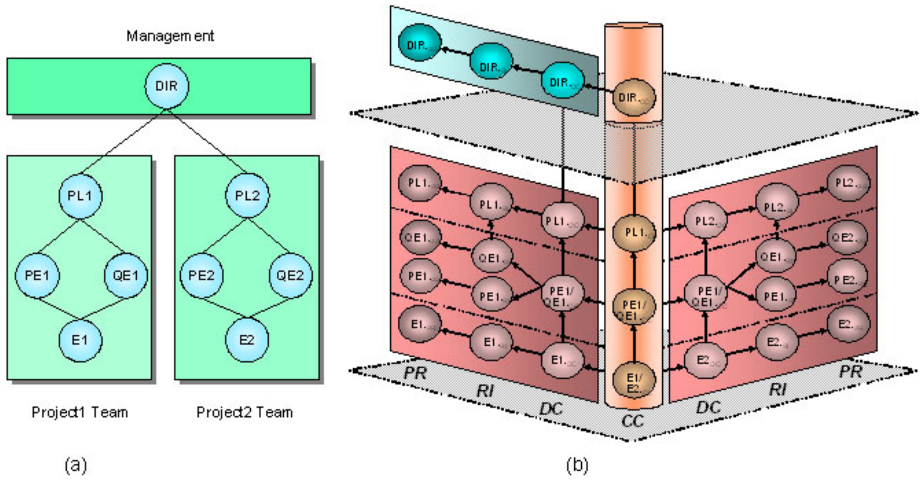


Fig. 3. Example of an organizational structure and its sub-role hierarchies

3.2 Separate Role Hierarchy Approach

In this section, we briefly review the shortcomings of the traditional role engineering process, in which a discrepancy between the role hierarchy and organization structure usually exists. Also, we propose another role engineering approach, in which role engineer constructs organizational and functional role hierarchies separately and associates each of roles in the hierarchies according to the organization's regulation.

3.2.1 Discrepancy Between Role Hierarchy and Organization Structure. Every organization has its own organization structure which consists of units such as departments, sections, or divisions etc. Furthermore, in each unit of the organization, there are job positions such as a *General Manager*, an *Assistant General Manager*, to perform their own tasks and responsibilities. Therefore, the organization's structure provides important authorization information to security analyzers in role engineering process.

But in most traditional role engineering process, since role engineer focuses on efficient permission management through permission inheritance characteristic, the constructed role hierarchy usually has different structure with that of the organization. With the role hierarchy which is inconsistent with the organization's structure, it is not easy for security administrator to perform security management tasks and reflect the change of security policy. In addition, considering a user assignment (UA) task of RBAC model is usually performed by employees of personnel department, the discrepancy between the role hierarchy and the organization's structure makes UA tasks complicated and error prone.

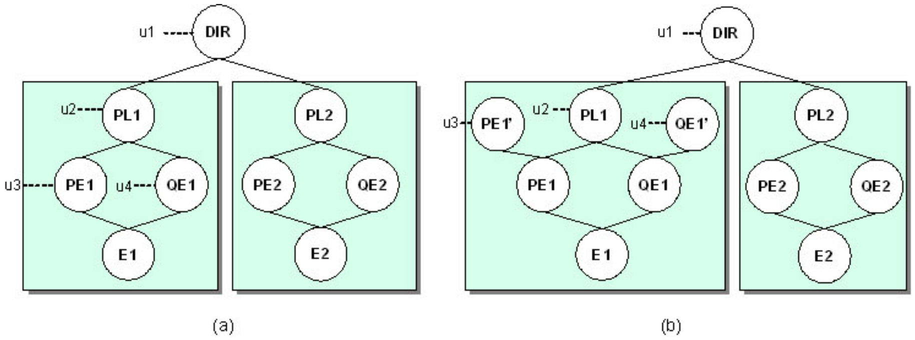


Fig. 4. Example of traditional role hierarchy

Figure 4 shows a typical role hierarchy which models two different properties of a single hierarchy; the permission inheritance feature and the organization structure. As a result, there is no clear distinction between the two concepts and then makes security management tasks difficult and unmanageable. For example, the structure like (i.e., the hierarchy among PE1, QE1, E1 roles) rarely can be found in real enterprise environment, since it is not common for an organization unit which has two different senior units. That kind of structure stems from the purpose of efficient permission management. In Figure 4, the permissions common to PE1 and QE1 are extracted and assigned to a new virtual role, E1, for efficient permission management. As mentioned earlier, it is desirable for personnel employees to perform user assignment tasks with the role hierarchy of which structure is as close as organization structure.

Furthermore, let us assume that roles PE1 and QE1 are mutually exclusive. In order to meet the separation of duty principle, the role PL1 can not be a common senior to both PE1 and QE1. While keeping the organization structure and satisfying the separation of duty principle, a role engineer need to add private roles of PE1 and QE1 (i.e. PE1' and AE1', respectively), which prevents conflicting permissions from inherited to their senior role. The resulting role hierarchy has more complex and different form than the original one.

All those problems are caused by different types of information, i.e. organization structure and permission inheritance, are mingled with a single role hierarchy without analyzing their own characteristics.

3.2.2 Separate Organizational Role Hierarchy and Functional Role Hierarchy. In the second role engineering approach, in order to solve the problems mentioned in 3.2.1, we use both organizational role hierarchy and functional role hierarchy. The organizational role hierarchy provides intuitive information on how the organization's units are related and which job positions exist and how they are inter-related etc., to non-security experts including employees in the personnel department. But the functional role hierarchy is constructed by role engineers through an analysis of permissions from the viewpoint of efficient permission management regardless of the organization's structure and the job positions.

The major advantage of this approach is an independent and transparent management of each structure. A personnel employee assigns a user (i.e. a new employee or a transferred employee) to a specific job position without the detailed knowledge of functional role hierarchies. A role engineer analyzes authorization information based on the organization's regulations and constructs the organizational role hierarchy. And then, he/she maps each job position to its corresponding organizational roles.

Figure 5 shows examples of organizational and functional role hierarchies. Comparing Figure 5 (a) with Figure 4 (a), no virtual role (i.e. E1, E2) is in organizational role hierarchy. Also, mutually exclusive roles (i.e. PE1 and QE1, PE2 and QE2) have the common senior roles in the organizational role hierarchy, but the conflicting permissions are prohibited from being inherited to senior roles (i.e. by assigning the conflicting permissions between PE1 and QE1 to roles r_2 and r_3 respectively in functional role hierarchies, roles PE1 and QE1 have the common senior role, PL1, in the organizational role hierarchy).

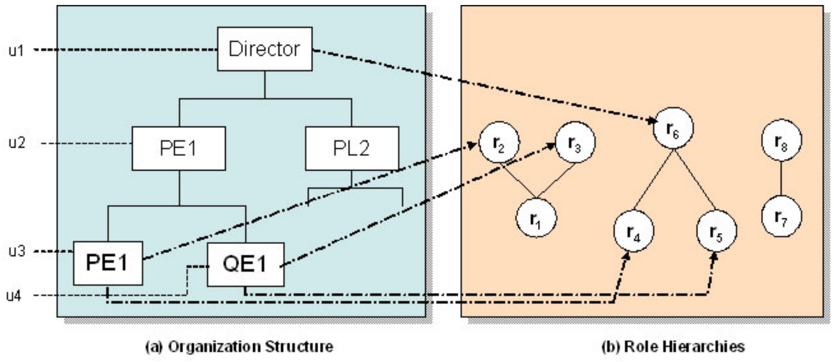


Fig. 5. Example of organizational and function role hierarchies

4 Conclusions and Future Work

Role engineering in RBAC system analysis and design phase means the process of defining roles, permissions, role hierarchies, constraints and assigning the permissions to the roles. It is the first step to implement RBAC system and also a crucial component of security engineering.

In this paper, we suggest two different approaches of role engineering methodology; *unified* and *separate* role hierarchy approaches. In the unified role engineering approach, roles are divided into sub-roles and a number of sub-role hierarchies are constructed based on the job functions and positions. After then, each sub-role hierarchies are interconnected to form a single hierarchy of the organization. Despite of its complex structure, the resulting role hierarchy provides an integrated view of role-based authorization structure.

On the contrary, in the separate role hierarchy approach, organizational and functional role hierarchies are constructed independently, and mapping roles between two hierarchies is performed. This enables role engineers to define the role hierarchy focusing on the efficient permission management and personnel employees to perform user assignment tasks transparent to the details of functional role hierarchies.

Our future work is to develop detailed processes for constructing unified role hierarchy, organizational and functional role hierarchies. In addition, the administration of each role hierarchy is another important research topic.

References

1. Ezedin Barka and Ravi Sandhu, “Framework for Role-Based Delegation Models," Proceedings of 23rd National Information Systems Security Conference, pp.101-114, Baltimore, Oct. 16-19, 2000.
2. Virgil Gligor, Serban Gavrila and David Ferraiolo, “On the Formal Definition of Separa-tion-of-Duty Policies and Their Composition," Proceedings of the IEEE Symposium on Se-curity and Privacy, pp.172-183, May 3-6, 1998.
3. Jonathan D. Moffett, “Control Principles and Role Hierarchies," 3rd ACM Workshop on Role Based access Control (RBAC). George Mason University, Fairfax, Virginia, USA. 1998.
4. Haio Roecle, Gerhard Schimpf, Rupert Weidinger, “Process-oriented approach for role-finding to implement role-based security administration in a large industrial organization," Proceedings of the fifth ACM workshop on Role-based access control, Germany, pp103-110, 2000.
5. HyungHyo Lee, YoungLok Lee, BongNam Noh, “A New Role-Based Delegation Model Using Sub-Role Hierarchies", International Symposium on Computer and Information Sci-ences (ISCIS 2003) LNCS 2869 pp.811-818, 2003.
6. Ravi Sandhu, Edward Coyne, Hal Feinstein and Charles Youman, “Role-based ac-cess control model," IEEE Computer, Volume 29, pp.38-47, Feb, 1996.
7. Lupu, E.C. and M.S. Sloman. “Reconciling Role-Based Management and Role-Based Ac-cess Control," in 2nd ACM Workshop on Role-Based Access Control. 1997.
8. Awischus, R. “Role Based Access Control with the Security Administration Manager," in 2nd ACM Workshop on Role-Based Access Control. 1997.
9. Gustaf Neumann, Mark Strembeck, “A Scenario-driven Role Engineering Process for Functional RBAC Roles," in SACMAT’02, pp.33-42, 2002.

An Efficient Pointer Protection Scheme to Defend Buffer Overflow Attacks

Yongsu Park[2] and Yookun Cho[1]

[1] Department of Computer Science and Engineering, Seoul National University
San 56-1 Shilim-Dong Gwanak-Ku, Seoul 151-742, Korea
cho@ssrnet.snu.ac.kr

[2] The College of Information and Communications, Hanyang University
17 Haengdang-dong, Seongdong-gu, Seoul 133-791, Korea
yspark@ssrnet.snu.ac.kr

Abstract. We present a new efficient pointer protection method to defend buffer overflow attacks. It uses a simple watermark to protect the pointer: during dereferencing the pointer variable, a watermark is also written/updated and before referencing the pointer variable, it verifies consistency of the watermark. If the pointer's watermark does not exist or was damaged, our scheme regards this as an intrusion and stops the execution. The proposed scheme has the following strong points. First, unlike other randomization methods, our scheme has no possibility of malfunction caused by the execution of arbitrary instructions. Second, we conducted various experiments on prototype implementation, which showed that our scheme is as secure as the previous randomization schemes. Third, experimental results showed that the performance degradation is not high. Forth, unlike other randomization schemes, our scheme can support attack profiling.

Keywords: system security, buffer overflow, randomization

1 Introduction

In spite of countless methods designed to cope with buffer overflow vulnerabilities, new attacks are continuously appeared such as format string attacks, heap overflows, or multiple free errors. Up till now, buffer overflows are still major cause of exploited vulnerability.

In order to address this problem, new approaches have been recently developed: code randomization, address randomization, and pointer randomization. Among them, code randomization and address randomization have demerits such as significant performance degradation or partial coverage of defense. PointGuard [3] that was appeared in USENIX Security 2003 uses pointer encryption/decryption to provide both efficiency and wide defense coverage together. However, the shortcoming is that when there exists an attack, PointGuard should execute arbitrary instructions, which causes possible malfunction of the privileged process.

In this paper, we present a new efficient pointer protection method to defend buffer overflow attacks. It uses a simple watermark to protect the pointer: during dereferencing the pointer variable, a watermark is also written/updated and before referencing the pointer variable, it verifies consistency of the watermark. If the pointer's watermark does not exist or was damaged, our scheme regards this as an intrusion and terminates the execution.

The proposed scheme has the following strong points. First, unlike other randomization schemes such as RISE [1], [6], and PointerGuard [3], our scheme has no possibility of malfunction caused by the execution of arbitrary instructions. Second, we conducted various experiments on prototype implementation, which showed that our scheme is as secure as the previous randomization schemes. Third, experimental results showed that the performance degradation is not high. Forth, unlike other randomization schemes, our scheme can support attack profiling.

Our scheme can be viewed as an approach to increase trustworthiness as well as to enhance security of the systems to be protected. The rest of this paper is organized as follows. In Section 2 we describe the previous work. In Section 3, we explain PointGuard and our method. In Sections 4 and 5, we analyze performance and security of our scheme. Finally, we offer some conclusions in Section 6.

2 Related Work

To defend buffer overflow attacks, there have been many research activities such as imposing run-time type checking [5,8] or enhancing the conventional programming languages [9]. As mentioned in Introduction, in spite of these methods new attacks are continuously appeared and buffer overflows are still major cause of exploited vulnerability. In this section, we deal with recently proposed approach which is so called “randomization”.

To the best of our knowledge, the first randomization scheme is ASLR (Address Space Layout Randomization) provided by the PAX project [11]. It randomly assigns the address of each memory object such as program code or data. Since the most attacks overflow a specific memory region to overwrite the adjacent memory objects, it seems that ASLR works effectively. Moreover, the execution overhead is very low (only $10{\sim}18\%$ slower). Bhatkar et al. [2] enhanced this concept and made a new implementation to cover wider range of defense. However, ASLR and [2] cannot defend GOT (Global Object Table) attack or function pointer attacks as will be shown in Section 4. Moreover, they suffer from address fragmentation.

Another approach is to randomize the instruction codes. [1,6] encrypts each instruction code in the program and when it is to be fetched, it is decrypted and then executed. This approach relies on the CPU emulator, which results in significant performance degradation.

Cowan presented the StackGuard that checks the integrity of activation records in the stack [4]. It is very efficient such that its performance degradation can be negligible. However, its coverage of defense is relatively narrow: it can only defend against the stack-related attacks.

To the best of our knowledge, the recently proposed PointerGuard [3] has both wide coverage of defense and efficiency together. We explain PointGuard and discuss its demerits in Sections 3 and 4.

3 Proposed Method

In Section 3.1, we explain PointGuard [3] and address some demerits. Then, we describe our method that relies on watermarking in Section 3.2.

3.1 PointGuard

PointGuard exploits the property that buffer overflow attacks modify function/data pointers. It operates as follows:

1. **Initialization:** first, symmetric key K is initialized as a random value, which is obtained by reading `/dev/urandom`. Then, K is stored in the protected page in the memory space.
2. **Dereference of the pointer:** whenever the pointer value P is required to be initialized or modified, P is encrypted with K and then the result is stored in the memory.
3. **Reference of the pointer:** after reading the encrypted value from the memory, it is deciphered by using K. Then, decrypted pointer value can be used.

The main idea of PointGuard is that by defending only the symmetric key K, attackers cannot guess all the pointer values. Moreover, illegally modified pointer values become meaningless since the decrypted addresses would have arbitrary values.

Pointer access occurs frequently so efficiency is very important. This is why authors use a single xor operation to implement encryption/decryption of the pointer.

Implementation of the PointGuard algorithm can be performed by the one of the following three ways [3]. First, the source code of the target process can be modified to support PointGuard. Second, we can modify the compiler to insert pointer encryption/decryption. Third, loader can be modified. Authors of [3] modified the gcc compiler to implement prototype version of PointGuard.

3.2 Proposed Scheme

One of the demerits of instruction/pointer randomization methods is that in the case of the attack arbitrary codes are executed. Considering that randomization methods are mainly applied to the privileged processes, execution of arbitrary codes could result in a serious disaster such as breaking important configuration files or causing system crashes.

The main idea of the proposed scheme is that we insert a watermark for each pointer to detect the modification of either the pointer or watermark before the reference.

1. **Initialization:** first, symmetric keys K_1 and K_2 are initialized as random values, which are from `/dev/urandom`. Then, they are stored in the protected page in the memory space.
2. **Dereference of the pointer:** whenever the pointer value P is required to be initialized or modified, P is encrypted by K_1 and then the result is stored in the memory. By another key K_2, P is encrypted and this value is inserted in the address P.

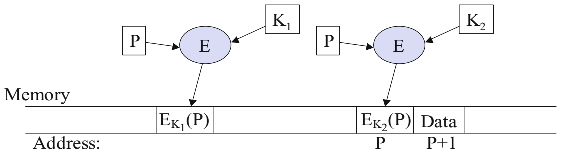


Fig. 1. Dereference of the pointer

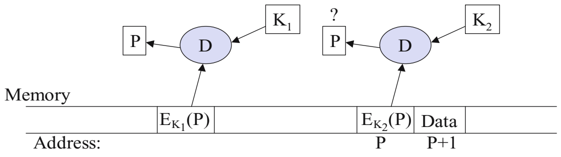


Fig. 2. Reference of the pointer

3. **Reference of the pointer:** after reading the encrypted value from the memory, it is deciphered by using K_1 to produce P. Then, the watermark in the address of P is decrypted by using K_2. If this value is identical to P, then, decrypted pointer value $(= P)$ can be safely used. Otherwise, the target process should be terminated.

As in PointGuard, we use a xor operation to encrypt/decrypt the pointer since performance degradation should be minimized. In this case, $P \oplus K_1$, $P \oplus K_2$ are stored in the memory with respect to P. If an attacker A reads these values, he can xor them to obtain $K_1 \oplus K_2$. If A replaces them with C' and $C' \oplus K_1 \oplus K_2$, he successfully forges the above check routine.

However, this attack is unlikely to occur since $P \oplus K_2$ is stored in the address P. If A knows $P \oplus K_2$, this means that he already knows P. Since we assume that K_1 and K_2 cannot be revealed (as in PointGuard), A is unable to know the value P by the property of the xor operation.

Another shortcoming of PointGuard is that it cannot protect all the pointers. More specifically, pointers invisible in the source code cannot be protected. Among them, we think that the most important pointers to be protected are frame pointer and return address in the stack since numerous buffer overflow attacks modify them. By inserting the watermark and the checking routine for them, defence coverage of the proposed scheme would be significantly extended.

4 Security Analysis

As mentioned in Section 3, the proposed scheme can be implemented by the one of the following three ways: source code modification, compiler modification, and loader modification. To simplify the implementation, we chose the first approach. We used C language and modified all the pointers in the source code to perform encryption / decryption and to check the watermarks.

To protect frame pointer/return address, we used modified gcc that has xor random canary method of StackGuard [4]. Although the algorithm is different, xor random canary method has the same object as that of our scheme for protecting frame pointer/return address in the stack.

Test programs consist of Straw man vulnerability overflowing example in Section 6 of [4] (overflowing function pointer), examples of Phrack 49 (typical buffer overflow and small buffer overflow) [10], and examples in Omega project (GOT attack) [7]. For all the tests, the programs were safely terminated and we were unable to find any success of attack in the proposed scheme.

Table 1 shows comparison of various buffer overflow defences. Compared schemes are PointGuard [3], PAX/ASLR [2], StackGuard [4], Bochs [6], and RISE [1].

Table 1. Comparison of buffer overflow defences

Methods	Execution of arbitrary codes	Defence against various attacks	
		Stack Smashing Attacks	Function Pointer Attacks
PAX/ASLR [2]	Yes	Yes/No	Yes/No
StackGuard [4]	No	Yes	No
PointGuard [3]	Yes	No	Yes
RISE [1]	Yes	Yes/No	Yes/No
Bochs [6]	Yes	Yes/No	Yes/No
Proposed Scheme	No	Yes	Yes

5 Performance Analysis

To evaluate the performance of the proposed scheme, we conducted the following three tests:

1. **Read:** repeat referencing the pointers in the pointer arrays.
2. **Write:** dereference the pointers by storing values in the pointer arrays.
3. **Read/Write:** repeat the job that reads a pointer value and then stores it in the another place.

Table 2 contains the performance results. Experimental environments are as follows: CPU, RAM, OS, and compiler are Intel P4/Xeon 1.8 GHz, 512 MBytes, Linux 2.6.4-1, and gcc 3.3.3, respectively. For each trial, we measured the elapsed time for 10^7 operations of the pointers. This table shows that our scheme is about 1.5$\sim$2 times slower for each pointer access. We think that performance degradation would be much less if we implement our scheme in the AST/RTL level of gcc, as in the experiment of PointGuard.

6 Conclusion

In this paper we presented a new efficient pointer protection method to defend buffer overflow attacks. It uses a simple watermark to protect the pointer: during dereferencing the pointer variable, a watermark is also written/updated and before referencing

Table 2. Comparison of elapsed time (in seconds)

Test	Unmodified source code	Proposed scheme
Read	17.9	34.1
Write	25.6	42.6
Read/Write	28.9	46.3

the pointer variable, it verifies consistency of the watermark. If the pointer's watermark does not exist or was damaged, our scheme regards this as an intrusion and stops the execution. The proposed scheme has the following strong points. First, unlike other randomization methods, our scheme has no possibility of malfunction caused by the execution of arbitrary instructions. Second, we conducted various experiments on prototype implementation, which showed that our scheme is as secure as the previous randomization schemes. Third, experimental results showed that the performance degradation is not high. Forth, unlike other randomization schemes, our scheme can support attack profiling. Our scheme can be viewed as an approach to increase trustworthiness as well as to enhance security of the systems to be protected.

References

1. Gabiela Barrantes, David H. Ackley, Stephanie Forrest, Trek S. Palmer, Darko Stefanovic, and Dino Dai Zovi. Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks. In *10th ACM Conference on Computer and Communication Security*, pages 281–289, October 2003.
2. Sandeep Bhatkar, Daniel C. DuVarney, and R. Sekar. Address Obfuscation: an Efficient Approach to Combat a Broad Range of Memory Error Exploits. In *12th USENIX Security Symposium*, pages 105–120, August 2003.
3. Crispin Cowan, Steve Beattie, John Johansen, and Perry Wagle. PointGuard: Protecting Pointers From Buffer Overflow Vulnerabilities. In *12th USENIX Security Symposium*, pages 91–104, August 2003.
4. Crispin Cowan, Calton Pu, Dave Maier, Jonathan Walpole, Peat Bakke, and Steve Beattie. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-overflow Attacks. In *7th USENIX Security Symposium*, pages 63–78, January 1998.
5. Richard Jones and Paul Kelly. Bounds Checking for C. avaliable at http://www-ala.doc.ic.ac.u/˜phjk/BoundsChecking.html, July 1995.
6. Gaurav S. Kc, Angelos D. Keromytis, and Vassilis Prevelakis. Countering Code-Injection Attacks With Instruction-Set Randomization. In *10th ACM Conference on Computer and Communication Security*, pages 272–280, October 2003.
7. Lamagra. Project OMEGA. avaliable at http://ouah.kernsh.org/omega1lam.txt.
8. Greg McGary. Bounds Checking for C and C++ Using Bounded Pointers. avaliable at http://gcc.gnu.org/projects/bp/main.html, 2000.
9. George C. Necula, Scott McPeak, and Westley Weimer. CCured: Type-Safe Retrofitting of Legacy Code. In *29th ACM Symposium on Principles of Programming Languates (POPL02)*, 2002.
10. Aleph One. Smasing The Stack For Fun And Profit. Phrack 49, File 14 of 16, 1996.
11. PaX team. The PaX Project. avaliable at http://pageexec.virtualave.net, 2001.

Parallel Hierarchical Radiosity: The PIT Approach

Fabrizio Baiardi, Paolo Mori, and Laura Ricci

Dipartimento di Informatica, Universitá di Pisa
Via F.Buonarroti, 56125 - Pisa, Italy
{ricci,baiardi,mori}@di.unipi.it

Abstract. Radiosity is a method to compute the global illumination of a scene. To reduce its complexity, hierarchical radiosity decomposes the scene into a hierarchy of patches and computes the light exchanged between patches at different levels, according to their distance and/or to the amount of light they emit. A distributed memory implementation of this method has been developed through *PIT*, a problem independent library that supports hierarchical applications on distributed memory architectures. *PIT* functions exploit a distributed version of the tree representing the hierarchical decomposition.

1 Introduction

Radiosity [5] defines the global illumination of a scene by decomposing objects into sets of patches and by computing the forms factor of each pair of patches, i.e. the fraction of the light they exchange. To reduce the overall complexity, a hierarchical approach exploits hierarchical patches and it computes the light exchanged between two patches at distinct levels, according to their distance and/or to the amount of light they emit. One of the problems posed by a parallel implementation of this method is the *irregular* and *dynamic* distribution of the patches in the scene. Furthermore, the computational loads of the patches may be very different because the light exchange is computed at distinct levels. This paper evaluates a parallel version of a hierarchical radiosity method developed through *PIT* [3,1,2], a problem independent programming library to support the development of irregular hierarchical applications on distributed memory architectures. Its key assumption is that both the sequential application and the parallel one can be built around a tree that represents the distribution of elements in the domain. The definition of element is problem dependant and it constitutes the elementary component of the considered problem. As an example, in the case of radiosity, an element is one patch, while in the case of the Barnes-Hut method it is a body. The interactions among elements are computed by applying proper operators to this tree. In the parallel version, the tree is mapped into the local memories of the processing nodes, pnodes. A new data type, the *Parallel Irregular Tree*, describes this mapping and it defines the operations to access information mapped onto other pnodes, to insert new nodes into the tree and to balance the computational load. To focus on the adoption of *PIT* to develop hierarchical radiosity on a distributed memory architecture, we have considered scenes in *flatland*, i.e. a bi-dimensional world. Sect. 2 describes the main functions of *PIT* . The sequential hierarchical approach to radiosity in flatland is revised in Sect. 3. Sect.

4 describes the parallelization of the method by *PIT*. Experimental results are discussed in Sect. 5.

2 The PIT Library

Parallel Irregular Tree, PIT, is a library addressed to the parallelization of irregular problems on distributed memory architectures. This library adopts a hierarchical method to solve irregular problems, and it is based upon the assumption that both the sequential and the parallel versions of a hierarchical method may be structured in terms of operations on a hierarchical tree, *Htree*. The *Htree* represents both the subdomains resulting from the hierarchical domain decomposition and the relation among them and any element they include. Any hierarchical method, to solve irregular problems, iteratively visits the *Htree* and it updates the properties of its elements. For each *hnode hn*, the method updates the properties of $element(hn)$, the element paired with *hn*, after collecting the neighbors of $element(hn)$. The operations to update the properties of an element depend upon the nature of the problem to be solved, *target problem*, and are denoted *target problem operators*, or simply operators. A goal of *PIT* is to preserve the code of these operators but, while in a sequential application the *Htree* is recorded in just one memory, in a distributed memory architecture the *HTree* is distributed across the local memories of all the pnodes. The functions of *PIT* support the application of these operators in spite of the mapping onto distinct memories. They are defined in terms of the *Parallel Hierarchical Tree*, *PITree*, the distributed version of the *Htree*. A *PITree* $P(H)$ is a tuple $\langle h_0, .., h_{np-1}, mht \rangle$, where np is the number of pnodes, that describes the mapping of a *Htree* H. Any h_i is the private *Htree* of P_i, the i-th pnode, i.e. the subset of all and only the hnodes of H that represent domains or elements mapped into the local memory of P_i. mht, instead, is the $mappingHtree$ that represents the hierarchical relations among $h_0, .., h_{np-1}$. mht includes the root R of H and all the hnodes on the paths from R to the root of any h_j. Each hnode of mht corresponds to a hnode hp of a private *Htree* h_j and it records j, the identifier of the pnode where hp has been mapped. mht is the minimal amount of information to be replicated in all the pnodes to enable the *PIT* functions to determine where any hnode of H has been mapped.

In the following, we focus on the functions of *PIT* used to define the parallel version of hierarchical radiosity methods. *PIT* is fully described in [3], together with the algorithm to map the *Htree* onto the pnodes. *PITree_Creation* is the function that partitions the initial set of elements among the *pnodes* and builds the *PITree*, i.e. both the private and the mapping *Htrees*. This function returns to each pnode a reference to the root of its private *Htree*. Then, this reference is trasmitted to the *PIT* functions that support the application of the target problem operators. To reduce the communication among the pnodes, the strategy to partition the elements takes into account the locality property. Since *PIT* is independent of the target problem and, in particular, of the element properties, the user has to define both a structure to record these properties and three functions, *include_el*, *decompose_el*, *remove_el*, to handle the elements. These functions are exploited by *PITree_Creation* to properly partition the elements.

Each pnode P_i applies the target problem operators to each hnode in its private Htree h_i, in a SPMD style. However, h_i does not include all the elements the operator requires, because some neighbours of a hnode hn of h_i, the remote neighbours of hn, may have been mapped onto distinct pnodes. *PITree_Completion* is the *PIT* function that collects all the elements required by an operator and creates, for each private *Htree* h_i, the *essential Htree* eh_i. eh_i is the union of h_i with all the remote neighbours of any element in h_i. In general case, *PITree_Completion* is invoked before the execution of each target problem operator so that the same operator of the sequential version of the application can be correctly applied to $eh_i \cap h_i$. The parameters of this function are the root of the private *Htree* and a user defined function *stencil* to compute the neighborhood stencil, i.e. the set of neighbors of each element. The *stencil* function depends upon the target problem operator so that distinct operators of the same problems may have distinct neighborhood stencils. To reduce the communication overhead, *PITree_Completion* implements a fault prevention strategy [1], where each *pnode* P_k determines, through the neighborhood stencil, which of its elements are required by P_h, $\forall h \neq k$ and sends these elements to P_h without an explicit request from P_h to P_k. *PITree_Balance* updates the mapping of the *PITree* onto the pnodes to balance the computational load, and it is invoked anytime the distribution of the elements in the domain is updated. Hence, in the simplest case, it is invoked after the execution of each operator. Its input parameters are a reference to the roots of the private *Htrees* and a threshold to prevent the recovery of very low unbalances. It returns a reference to the root of the updated private *Htree*.

3 Hierarchical Radiosity in Flatland

Hierarchical radiosity has been introduced to compute the heat transfer between surfaces in [9]. It has been applied in computer graphics to compute the global illumination of unoccluded environments in a scene in [6] and to compute the illumination of scenes with occlusions in [4]. Hierarchical radiosity [7] discretizes the objects of the scene through small elements, denoted as *patches*, whose brightness is assumed to be constant. The radiosity B_i of a patch p_i depends upon those of all the other patches p_j according to the equation:

$$B_i A_i = E_i A_i + \rho_i \sum_{j=0}^{n-1} B_j A_j F_{j,i}$$

where A_i is the area of p_i, E_i is its emissivity, i.e. the light emitted by p_i, ρ_i is its reflectance, and $F_{j,i}$ is the percentage of light leaving the patch p_j that reaches p_i. $F_{i,j}$ is the *form factor* between p_j and p_i.

In *flatland* [8,10], the bidimensional world we consider, a scene is a set of polygons. Each polygon is decomposed into a set of segments, each with two sides. Only one side interacts with the other polygons, because the other one is internal to the polygon. Let us consider a pair of segments S_i and S_j. A popular and efficient formula for the form factor $F_{i,j}$ is the *the string rule* [9] defined in terms of the *strings* stretched from the endpoints of S_i to the corresponding endpoints of S_j, i.e. the *uncrossed strings*, and those that links up the opposite endpoints, i.e. the *crossed strings*. $F_{i,j}$ is proportional to the difference between the lengths of the crossed strings and that of the uncrossed ones. If some parts of S_i and S_j are not mutually visible, the strings are "stretched" around the obstruction so that the string rule can consider an arbitrary number of occluding segments.

Hierarchical radiosity recursively partitions each segment into a hierarchy of patches and it computes the interactions among patches so that a segment interacts with the other ones at distinct levels. Approximated interactions are computed among low level patches, while accurate interactions are computed among high level ones. An *interaction list* paired with each patch p_i records the interacting patches of the other segments. By pairing each element of this list with the list of the related occlusions, we produce the visibility list that includes the information to compute the radiosity of the corresponding patch.

Hierarchical radiosity computes a sequence of approximations starting from a domain that includes all the segments of the polygons in the scene. The algorithm computes the visibility list of each segment and it sets the initial radiosity of each segment to its emissivity. Then, an iterative procedure computes the global radiosity of each segment. Each iteration applies the functions *Gather*, *PushPull* and *RefineLink*. For each segment S_i, *Gather* collects the light emitted by the other segments that reaches S_i, according to the current decomposition. The function is recursively applied to S_i and to any patch produced by the decomposition of S_i and it applies the *string rule* to compute the light contributions received by any patch in the visibility list. For each segment S_i, *PushPull* redistributes among the patches of S_i the light contributions received at all the levels of the hierarchy as computed by *Gather*. *PushPull* includes two steps. The *Push* step pushes the radiosity of all the patches of S_i down into the hierarchy, from the root patch that represent the segment S_i to the leaf ones. The radiosity of a patch p_i is updated by the current value of the radiosity of the patch p_i belongs to. The *Pull* step pulls the radiosity of each patch of S_i up in the hierarchy, from the leaf patches to the root one. The radiosity of p_i is the weighted average of those of all the subpatches of p_i. *PushPull* computes the overall radiosity of the scene as well and the method terminates if the difference between the current value and the one at the previous iteration is smaller than a user defined threshold. *RefineLink* partitions a patch p_i because its interactions have to be computed with a better accuracy, i.e. because its radiosity is larger than a predefined threshold. The partition is defined according to the geometric properties of p_i. *RefineLink* updates the visibility lists of p_i and of each patch p_j in the interaction list of p_i.

4 Parallelizing Hierarchical Radiosity: The PIT Approach

In general, radiosity methods represents distinct objects of the domain by distinct trees, where the nodes of the tree represent the patches. In our approach, instead, a single *Htree* represents the whole domain and it includes all the objects. The *hnodes* represent the spaces that include the patches. Fig. 1 shows the *Htree* and the domain it represents.

The scene includes two segments, S_0 and S_1, partitioned into patches. The white hnodes of the *HTree* represent the spaces of the domain, while the black ones represent the elements, i.e. the patches. A node corresponding to a patch stores the patch properties, i.e. the coordinates of the segment, its orientation with respect to the polygon that includes it, the emissivity, the reflection and the current radiosity.

The initial domain is partitioned until each space includes one element only. For instance, the segment S_1 of Fig. 1 has been partitioned into two segments, p_0 and p_1, at level $l = 1$, because the smallest space that includes it, that is the space that represents

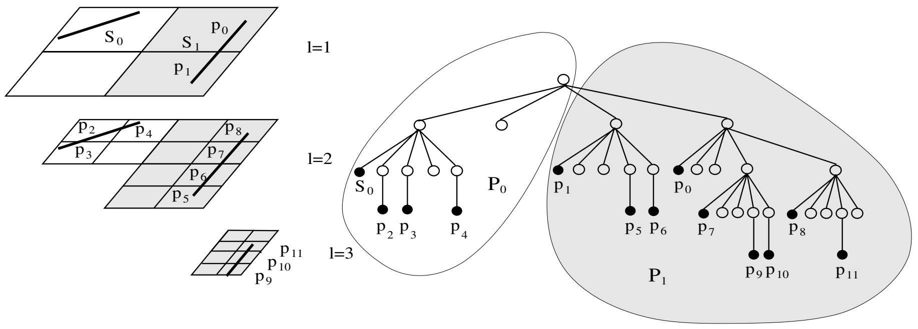


Fig. 1. Domain decomposition and PITree

the whole domain, also includes the segment S_0. This agrees with the philosophy of this method that partitions a segment S_i before the computation if another segment S_j belongs to the smallest space including S_i. In this case, S_i and S_j are close in the domain, and they will be decomposed anyway by the refinement function to compute their interactions. Fig. 1 shows a simple domain decomposition and the related *PITree*.

Fig. 2 describes the parallel version of the hierarchical radiosity method developed through the PIT library by inserting the proper *PIT* functions into the sequential code. The first *PIT* function invoked by the parallel code is *PITree_creation* that, starting from the initial set of elements of the domain, creates the *PITree* and returns to each pnode a reference to its *private HTree*. As an example, in Fig. 1 the segment S_0 and all its patches have been mapped onto the pnode P_0, while S_1 and all its patches have been mapped onto P_1. Before the first iteration, *PITree_completion* gathers the information to compute the visibility list of each segment. In this case, the neighborhood stencil of a segment S_i, i.e. *vis_stencil*, includes all the spaces that are visible from the external side of S_i. Since the radiosity of a patch p_i depends upon those of other patches in the scene, some of the patches that pnode P_a needs to compute the radiosity of p_i may have been mapped onto another pnode P_b $a \neq b$, *PITree_completion* is applied before each operator that requires the remote neighbours of a patch. *Gather* and *PushPull* compute the radiosity of a local patch in terms of that of local and of remote ones, while *RefineLink* requires local values only. However, if a patch and all the subpatches it has been recursively partitioned into are mapped onto the same pnode, also *PushPull* requires local values only. The patches that *Gather* needs to compute the contributions to the radiosity of p_i are a function of the visibility list of p_i. This list includes the references to all the patches that interact with p_i and to all those that occlude these interactions. To collect the remote neighbors *Gather* requires, the fault prevention strategy can be applied. In the parallel version of the method the two steps of *PushPull* are implemented by two distinct procedures, *Push* and *Pull*. The neighborhood stencil of *PushPull* is very simple because, in the *Push* step, the stencil of p_i includes only the patches including p_i. In the *Pull* step the stencil of p_i includes any patch resulting from the decomposition of p_i . Furthermore, both procedures are implemented through a breadth first visit of the tree. The adoption of an anticipated visit is not convenient because the execution of the *Push* step and/or the *Pull* one on the objects whose patches are not mapped onto the same pnode strongly serializes the overall computation.

```
hierarchical_radiosity_method(element_list *scene) {
    pht_root = PITree_creation(scene, include_el, decompose_el, remove_el);
    PITree_completion(vis_stencil, all_levels);
    Visib_list_determ_H(pht_root);
    while (not end) {
        PITree_completion(interaction_list, all_levels);
        Gather(pht_root)
        for level from L_min to L_max
            PITree_completion(push_stencil, level);
            Push(pht_root, level)
        for level from L_max to L_min
            PITree_completion(pull_stencil, level);
            Pull(pht_root, level)
        end = RefineLink_H(pht_root);
        pht_root = PITree_balance(pht_root); }}
```

Fig. 2. Hierarchical Radiosity PIT Parallel Code

RefineLink inserts into the *HTree* a new *hnode* for each new patch of the scene. Since the new patches created by a pnode P_i partition the patches already mapped onto P_i, they are mapped onto P_i as well. This is coherent with the adopted methodology because, if the *PIT* strategy maps a space A onto a pnode P_i, then maps onto P_i all the subspaces of A before the other spaces of the domain. To determine the computational load of each element, we observe that most of the computation is devoted to compute the form factors. Hence, $l(p_i)$, the load of a patch p_i, can be estimated in terms of the number of interactions executed by p_i that is proportional to the size of the visibility list of p_i. *RefineLink* updates the load of the pnodes, because it partitions the existing patches and the new patches generate new interactions. The partitioning of p_i affects both the visibility list of p_i and that of any patch p_j that interacts with p_i. This decreases $l(p_i)$, because the interaction with p_j has been removed from its visibility list, and it increases $l(p_j)$, because the interaction with p_i has been replaced by those with the subpatches of p_i. Moreover, new elements, i.e. the subpatches of p_i, have been added to the domain . The initial load of these elements depend upon the visibility list they inherit from p_i. *PITree_balance* is invoked after *RefineLink* to recover any unbalance in the computational load.

5 Experimental Results

To evaluate both the effectiveness and the performances of the PIT approach, this section presents some experimental results of the PIT parallel version of the hierarchical radiosity method. The first scene we consider is very simple and it is shown in Fig. 3. Initially it includes 224 segments that compose 48 polygons.

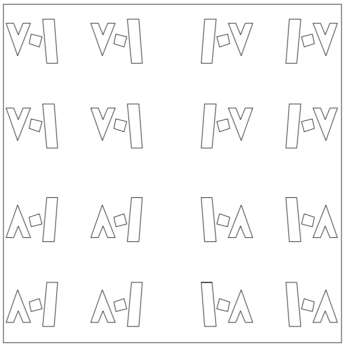


Fig. 3. Test scene: 224 segments

After 5 iterations only of the method, the total number of patches of the scene is about 3500% of the initial one. Fig. 4 shows the subdomain assigned to pnode 0 after 5 iterations in an the case of an execution on 10 pnodes.

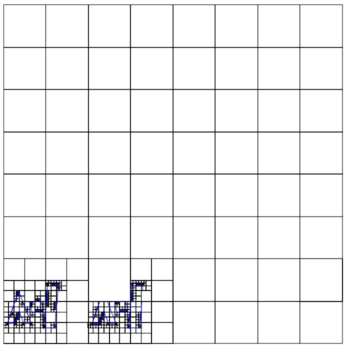


Fig. 4. Subdomain of pnode 0 after 5 iterations

The hierarchical decomposition of the domain is highly irregular, because some segments, or part of it, have been decomposed more deeply than others. We recall that this decomposition depends upon both the geometrical features of the scene and the exchange of light among the segments.

The first parallel architecture we consider in our experiments is a cluster of 10 workstations. Each workstation is a PC with an Intel Pentium II CPU (266 MHz) and 256 Mbyte of local memory and the interconnection network is a switched 100Mbit Fast Ethernet.

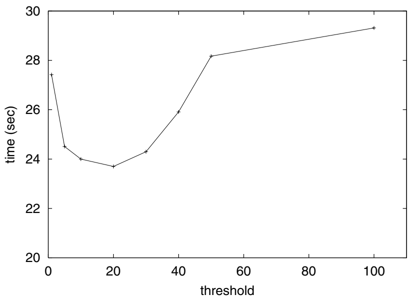


Fig. 5. Completion time with alternative load balancing thresholds

The first set of experimental results concerns the importance of the load balancing to achieve a good efficiency. Fig. 5 shows the completion time of 30 iterations of hierarchical radiosity when applied to the scene of Fig. 3 for different values of the load balance threshold. We can notice that the lowest completion time is achieved by adopting a balance threshold of 20%. In this case, the largest unbalance that is tolerated by the parallel application is 20% of the average load of the pnodes. Moreover, the figure shows that, by adopting a load balance threshold of 100%, the completion time is 116% of the optimal one. Hence, considerable performance improvements can be achieved by updating the data mapping at run time to recover load unbalances. Moreover, this also confirms the effectiveness of the PIT approach. The next set of experimental results evaluates the performance of the parallel implementation. Fig. 6 shows the efficiency of the hierarchical radiosity method for a variable number of pnodes. The scene considered in this experiment is that of Fig. 3, and 30 iterations have been executed.

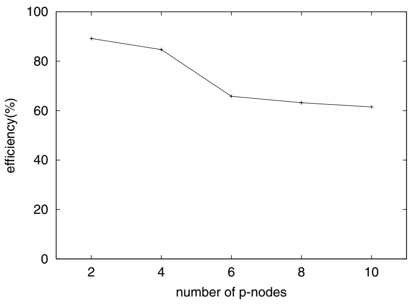


Fig. 6. Efficiency for the Hierarchical Radiosity: 224 segments

The figure shows that, even on 10 pnodes, our implementation achieves an efficiency close to 60%. The efficiency improves for a lower number of pnodes. For instance, it is about 80% in the case of 4 pnodes. A simple way to increase the efficiency is by considering larger scenes. Fig. 7 shows the other scene considered in our experiments that includes 896 segments that defines 192 polygons.

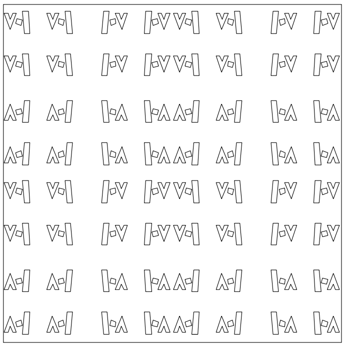


Fig. 7. Test scene: 896 segments

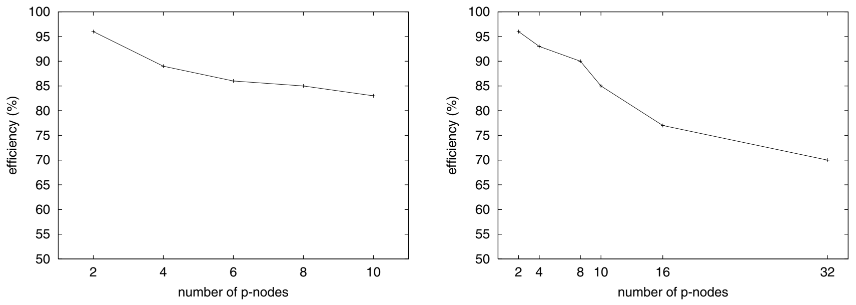


Fig. 8. Efficiency for Hierarchical Radiosity: 896 segments

Fig.8 shows the performance of the *PIT* hierarchical radiosity method, on two architectures. The left size of the figure shows the experiments executed on the architecture previously described and resulting in an efficiency larger than 80%, on 10 pnodes. Such a good efficiency is mostly due to the high locality of the scene. In fact, the large number of polygons of the scene results in a large number of occlusions among segments. In turns, this implies that each segment mainly interacts with close segments. The mapping strategy of *PIT* exploits at best this locality to minimize the communication overhead.

We have also considered an IBM Linux Cluster including 64 nodes, each with 2 Intel Pentium III (1.133 GHz) and 1Gbyte of local memory. The interconnection network is a Miricom LAN “C" Version. The right side of Fig. 8 shows the performance on this architecture. These results confirm the previous ones because the efficiency is larger than 70% even on 32 pnodes.

References

1. F. Baiardi, S. Chiti, P. Mori, and L. Ricci. Parallelization of irregular problems based on hierarchical domain representation. *Proceedings of HPCN 2000: Lecture Notes in Computer Science*, 1823:71–80, May 2000.

2. F. Baiardi, S. Chiti, P. Mori, and L. Ricci. Integrating load balancing and locality in the parallelization of irregular problems. *Future Generation Computer Systems: Elsevier Science*, 17:969–975, June 2001.
3. F. Baiardi, P. Mori, and L. Ricci. Pit: A library for the parallelization of irregular problems. *Proceedings of the 6th International Conference on Applied Parallel Computing (PARA 2002): Lecture Notes in Computer Science*, 2367:185–194, 2002.
4. M.F. Cohen and D.P. Greenberg. The hemi-cube: a radiosity approach for complex environment. *Computer Graphics*, 19(3):31–40, 1985.
5. C.Puech F.X.Sillion. *Radiosity and Global Illumination*. Kaufmann Publ., 1994.
6. C.M. Goral, K.E. Torrance, D.P. Greenberg, and B. Battaill. Modelling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):213–222, 1984.
7. P. Hanrahan, D. Salzman, and L. Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics*, 25(4):197–206, 1991.
8. P.S. Heckbert. Radiosity in flatland. *Eurographics 92*, 11(3):181–192, 1992.
9. H.C. Hottel. *Radiant heat transmission*, chapter 4. McGraw Hill, 1954.
10. R. Orti, S. Riviére, F. Durand, and C. Puech. Radiosity for dynamic scenes in flatland with the visibility complex. *Computer Graphics Forum*, 15(3):237–248, 1996.

Optimizing Locationing of Multiple Masters for Master-Worker Grid Applications

Cyril Banino

Norwegian University of Science and Technology (NTNU)
Department of Computer and Information Science (IDI)
Sem Sælandsvei 7-9 NO-7491 Trondheim, Norway
Cyril.Banino@idi.ntnu.no

Abstract. The problem of allocating a large number of independent tasks to a heterogeneous computing platform is considered. A non oriented graph is used to model a Grid, where resources can have different speeds of computation and communication. We claim that the use of multiple masters is necessary to achieve good performance on large-scale platforms. The problem considered is to find the most profitable master locations in order to optimize the platform throughput.

1 Introduction

A recent trend in high performance computing is to deploy computing platforms that span over large networks in order to harness geographically distributed computing resources. The aim is often to provide computing power to applications at unprecedented scale. Good candidates for such environments are *Master-Worker* applications, composed of a large number of computational tasks independent from each other, i.e. where no inter-task communications take place, and where the tasks can be computed in any order. Many applications have been and can be implemented under the Master-Worker paradigm. They include: Processing of large measurement data sets like the SETI@home project [1], biological sequence comparisons [2], or also distributed problems organized by companies like Entropia [3].

The Master-Worker paradigm consists in the execution of independent tasks by a set of processors, called *workers*, under the supervision of a particular processor, the *master*. The master holds all the tasks initially, and sends them out to the workers over a network. Workers compute their tasks, and send the results of the computation back to the master. This scheduling problem is well recognized, and several studies [4,5] have recently revisited the Master-Worker paradigm for clusters and Grids. Following previous studies [4,5], the performance measure adopted in this paper is the steady-state throughput of the platform, i.e. the number of tasks computed per time unit.

This work does not directly address the dynamically changing nature of large-scale computing platforms. Although this is an important consideration, this paper nonetheless suggests a simple but mandatory departure from traditional implementations: The need to deploy several masters to efficiently utilize currently emerging large-scale platforms.

Multiple masters are needed since the centralization of the tasks in one single place, not only makes the system more vulnerable to failures [6], but also clearly limits the scalability of the system. In this paper, we propose to circumvent the master bottleneck by introducing multiple masters. The problem becomes to determine how many masters should be used, and where should these masters be located on the platform graph in order to achieve better performance. As a consequence, a discrete network location problem arises, and this paper contributes insights on the problem difficulty within static settings. Static knowledge cannot be underestimated, as after all, a dynamic context may often be viewed as a succession of static contexts. This work hence provides an important first step for efficient deployments of large Master-Worker applications on computational Grids.

Growing the number of control resources to unblock the serial bottleneck has been successful in many different domains: Multi-mastering is employed to allow simultaneous bus transfers between peripherals within a single system, multi-server architectures are used for massively multi-player network games, replication in Data Grid environments ensure efficient access to widely distributed data [7]. Closely related to our work is the hierarchical Master-Worker implementation proposed by Aida et al. [8] where a supervisor controls multiple processor sets, each of which is composed of a master and several workers. The supervisor achieves load balancing by migrating tasks among masters. However, the work of Aida et al. does not take in consideration the topology of the platform, an essential aspect for large-scale Grids that we address in this paper.

To the best of our knowledge, Shao et al. [5] were the first to consider resource selection problems within the steady-state Master-Worker scheduling framework. The authors aim at selecting performance-efficient hosts for both the master and worker nodes. For that end, an exhaustive search is performed, consisting in solving n network-flow problems, where n is the number of processors composing the platform. Then the configuration that achieved the highest throughput is selected. Unfortunately, this approach is not applicable with several masters. There are indeed $\binom{n}{p}$ possible master locations sets, where p is the number of masters to be located on the platform. For this reason, we cannot simply compute the best scheduling strategy for each set, and then select the best result. As an example, for $n = 50$ and $p = 10$, a problem that is not large, the resulting number of possibilities is $10,272,278,170$. Clearly, even for moderate values of n and p, such enumeration is not realistic, and we need more advanced techniques. We introduce a cost model to establish and operate masters on the platform graph. Given a general interconnection graph, we consider the problem of selecting a master locations set that optimizes the throughput of the platform within a budget constraint B. We term such problem the *B-COVER* problem.

The rest of this paper is organized as follows: Related work is exposed in Section 2. Our model of computation and communication is introduced in Section 3. The *B-COVER* problem is formally stated and shown to be NP-hard in Section 4. We give a simple heuristic in Section 5, and present our experimental results in Section 6. Finally, conclusions and future work are discussed in Section 7. The extended version of this paper [9] introduces another discrete location problem, whose goal is to utilize all the computing resources of the platform at minimal cost. Possible extensions of our model, as well as a wide set of simulation results are also presented in [9].

2 Related Problems

The *B-COVER* problem is related to the *Facility Location* class of problems [10,11]. A classic facility location problem is a spatial resource allocation problem in which one or more service facilities have to be located to serve a geographically distributed set of population demands according to some objective function. The term facility is used in its broadest sense, as it is meant to include entities such as air and maritime ports, factories, warehouses, schools, hospitals, subway stations, satellites, etc [10]. In our case, we can assimilate the facilities to the master processors, the service to the input files, and the set of demands to the computing power of the worker processors. Of particular interest because close in spirit to the *B-COVER* problem, are the *Maximal Covering Location Problem* [12] that addresses planning situations which have an upper limit P on the facilities to be located, and the *Fixed Charge Location Problem* [10] that introduces capacities as well as costs constraints on the facilities to be located.

However, the *B-COVER* problem slightly differs from traditional facility location problems since the tasks allocated to workers produce some results that must be gathered at master location sites. In fact, the *B-COVER* problem can be expressed as a *Supply Chain Management* problem [13]. A supply chain can be defined as a network of facilities that manufactures finished products, and distributes these products to customers. Supply chain management involves deciding (i) where to produce, what to produce, and how much to produce at each site, and (ii) where to locate plants and distribution centers [13]. In our case, we can assimilate the manufactured products to output files and the plants to worker processors. The master processors will play two roles: Supply raw material (i.e. input files) to the plants, and then collect and distribute the final products to the user.

3 Multiple Master Model

Our model builds on the model presented by Banino et. al [14], but differs in that we here separate the link bandwidths from the communication times between processors, as well as by differentiating input files from output files. We also introduce a model for the master locations selection.

The target architectural framework is represented by a graph $G = (V, E)$ as illustrated in Fig. 1. Each vertex $P_i \in V$ represents a computing resource of weight w_i, meaning that processor P_i requires w_i units of time to process one task. Each edge $e_{i,j}$: $P_i \rightarrow P_j$ represents a communicating resource having a bandwidth equal to $\gamma_{i,j}$, which limits the amount of data that can be transfered on link $e_{i,j}$ per time unit in both direction.

The target application tasks are modeled as requiring some input data file of size β_I, and producing some output data file of size β_O. It takes $c_{i,j}$ time units to transfer one input file from processor P_i to a neighbor processor P_j, and $c'_{i,j}$ time units to transfer one output file from P_i to P_j. The communication times $c_{i,j}$ and $c_{j,i}$ can possibly be different, due to, say different I/O hardware device of processors P_i and P_j. This holds also for $c'_{i,j}$ and $c'_{j,i}$. Computations and communications are supposed to be atomic, i.e these operations, once initiated, cannot be preempted.

All w_i are assumed to be positive rational numbers since they represent the processor computing times. We disallow $w_i = 0$ since it would permit processor P_i to perform

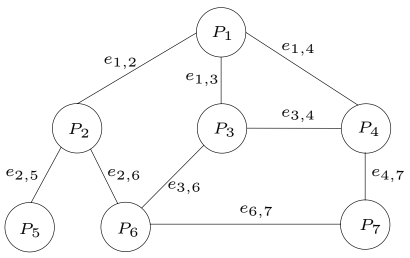


Fig. 1. Grid graph. Vertices and edges stand for processors and communication links

an infinite number of tasks, but we allow $w_i = +\infty$; then, P_i has no computing power, but can still forward tasks to other processors (e.g. to model a switch). Similarly, we assume that all $c_{i,j}$ and $c'_{i,j}$ are positive rational numbers since they correspond to the communication times between two processors.

The *full overlap, single-port* model [14] is adopted, in order to represent the operation mode of the processors. Under this model, a processor can simultaneously receive data from one of its neighbors, perform some (independent) computation, and send data to one of its neighbors. At any given time-step, a given processor may open only two connections, one in emission and one in reception. Stating the communication model more precisely: If P_i sends an input file to P_j at time-step t, then: P_j cannot start executing the associated task, or forwarding this input file before time-step $t + c_{i,j}$; P_j cannot initiate a new receive operation before time-step $t+c_{i,j}$ (but, it can perform a send operation and independent computation). P_i cannot initiate another send operation before time-step $t + c_{i,j}$ (but, it can perform a receive operation and independent computation). This model holds also for the communication of output files.

Finally, let $J_m \subseteq V$ denote the index set of the processors susceptible to be chosen as a master, and for each processor $P_i \in J_m$, let $x_i \in \{0,1\}$ be the decision variable to place a master at location P_i, i.e. $x_i = 1$ if P_i is chosen as a master, and $x_i = 0$ otherwise. Let f_i be the fixed cost of establishing a master at location P_i, and t_i be the per task cost of operating a master at location P_i. All f_i and t_i are assumed to be positive constants since, from a practical viewpoint, it is rather absurd to have negative costs for establishing master locations.

4 The *B-COVER* Problem

4.1 Mathematical Formulation of *B-COVER*

To formally define the *B-COVER* problem, let $n(i)$ denote the index set of the neighbors of processor P_i. During one time unit, α_i is the fraction of time spent by P_i computing, $s_{i,j}$ and $s'_{i,j}$ are the fractions of time spent by P_i sending input and output files to its neighbor $P_j \in n(i)$, $r_{i,j}$ and $r'_{i,j}$ are the fractions of time spent by P_i receiving input and output files from its neighbor $P_j \in n(i)$. Defined on a platform graph G, a mathematical formulation of the *B-COVER* problem can be stated by the following integer linear program, whose objective is to maximize the throughput $n_{task}(G)$ of the platform graph G.

Maximize

$$n_{task}(G) = \sum_{i \in V} \frac{\alpha_i}{w_i},$$

Subject to

$$\left\{\begin{array}{l|l}
(1)\ \forall i,\ 0 \le \alpha_i \le 1 & (12)\ \forall i \in J_m,\ x_i + \sum_{j \in n(i)} r_{i,j} \le 1 \\
(2)\ \forall i, \forall j \in n(i),\ 0 \le s_{i,j} \le 1 & (13)\ \forall e_{i,j} \in E,\ \left(\frac{s_{i,j}}{c_{i,j}} + \frac{r_{i,j}}{c_{j,i}}\right)\beta_I + \\
(3)\ \forall i, \forall j \in n(i),\ 0 \le s'_{i,j} \le 1 & \qquad \left(\frac{s'_{i,j}}{c'_{i,j}} + \frac{r'_{i,j}}{c'_{j,i}}\right)\beta_O \le \gamma_{i,j} \\
(4)\ \forall i, \forall j \in n(i),\ 0 \le r_{i,j} \le 1 & (14)\ \forall i \notin J_m,\ g_i = 0 \\
(5)\ \forall i, \forall j \in n(i),\ 0 \le r'_{i,j} \le 1 & (15)\ \forall i \in J_m,\ 0 \le g_i \le \left(\frac{1}{w_i} + \mu_i\right) \times x_i \\
(6)\ \forall i \in J_m,\ x_i \in \{0,1\} & (16)\ \forall i \notin J_m,\ g'_i = 0 \\
(7)\ \forall i, \forall j \in n(i),\ s_{i,j} = r_{j,i} & (17)\ \forall i \in J_m,\ 0 \le g'_i \le \left(\frac{1}{w_i} + \mu'_i\right) \times x_i \\
(8)\ \forall i, \forall j \in n(i),\ s'_{i,j} = r'_{j,i} & (18)\ \forall i,\ g_i + \sum_{j \in n(i)} \frac{r_{i,j}}{c_{j,i}} = \frac{\alpha_i}{w_i} + \sum_{j \in n(i)} \frac{s_{i,j}}{c_{i,j}} \\
(9)\ \forall i,\ \sum_{j \in n(i)} (s_{i,j} + s'_{i,j}) \le 1 & (19)\ \forall i,\ g'_i + \sum_{j \in n(i)} \frac{s'_{i,j}}{c'_{i,j}} = \frac{\alpha_i}{w_i} + \sum_{j \in n(i)} \frac{r'_{i,j}}{c'_{j,i}} \\
(10)\ \forall i \in J_m,\ x_i + \sum_{j \in n(i)} s'_{i,j} \le 1 & (20)\ \sum_{i \in J_m} (f_i x_i + t_i g_i) \le B \\
(11)\ \forall i,\ \sum_{j \in n(i)} (r_{i,j} + r'_{i,j}) \le 1 &
\end{array}\right.$$

• Equations (1), (2), (3), (4) and (5) specify that all the activity variables must belong to the interval $[0, 1]$, as they correspond to the fraction of activity during one time unit.

• Equation (6) specifies candidate locations for establishing the masters.

• Equations (7) and (8) ensure communication consistency: The time spent by P_i to send input (output) files to P_j should be equal to the time spent by P_j to receive these input (output) files from P_i.

• Equation (9) ensures that send operations to neighbors of P_i are sequential.

• Equation (10) enforces that masters do not send output files to other processors.

• Equation (11) ensures that receive operations to neighbors of P_i are sequential.

• Equation (12) enforces that masters do not receive input files from other processors.

• Equation (13) ensures that link bandwidths cannot be exceeded. This constraint is due to our hypothesis that the same link $e_{i,j}$ may be used in both directions simultaneously.

• Equations (14) and (15) specify that only the masters can generate input files to be allocated on the Grid. Let g_i be the number of input files generated by P_i per time unit. In practice g_i will be limited by the number of tasks that P_i can process per time unit (i.e. $\frac{1}{w_i}$) plus the maximum number of input files μ_i that P_i can communicate to its neighbors per time unit. Under the base model, μ_i is bounded by the inverse of the smallest communication time $c_{i,j}$ of the neighbors of P_i. We hence have $\mu_i = \frac{1}{\min\{c_{i,j} | j \in n(i)\}}$.

• Equations (16) and (17) specify that only masters can collect output files generated on the Grid. Let g'_i be the number of output files collected by P_i per time unit, and μ'_i

be the maximum number of output files that P_i can receive from its neighbors per time unit. We hence have $\mu'_i = \frac{1}{\min\{c'_{j,i} | j \in n(i)\}}$.

• Equations (18) and (19) stand for conservation laws. For every processor P_i, the number of input files generated, plus the number of input files received, should be equal to the number of input files processed, plus the number of input files sent (equation (18)). Similarly, for every processor P_i, the number of output files collected, plus the number of output files sent, should be equal to the number of input files processed, plus the number of output files received (equation (19)).

It is important to understand that equations (18) and (19) really apply to the steady-state. Consider an initialization phase during which some input files are forwarded to processors, but no computation is performed. Then some computation take place, and some output files are produced. At the end of this initialization phase, we enter the steady-state: During each time-period in steady-state, each processor can send/receive data, and simultaneously perform some computation.

Equations (7), (8), (18) and (19) ensure that there are as many input files produced as output files consumed. Indeed, by combining these four equations, we can obtain the following equation:$\sum_{i \in J_m} g_i = \sum_{i \in J_m} g'_i$

• Equation (20) ensures that the costs generated by establishing (f_i) and operating (t_i) the chosen master locations do not exceed the budget constraint B.

• Finally, the objective function is the number of tasks computed within one unit of time, i.e. the platform throughput $\sum_{i \in V} \frac{\alpha_i}{w_i}$.

4.2 Complexity of *B-COVER*

Theorem 1. B-COVER *is NP-hard.*

Proof. We reduce the *MAXIMUM KNAPSACK* (MK) problem [15] to the *B-COVER* problem.

MAXIMUM KNAPSACK

INSTANCE: Finite set U, for each $u \in U$ a size $s(u) \in \mathbf{Z}^+$ and a value $v(u) \in \mathbf{Z}^+$, a positive integer $B \in \mathbf{Z}^+$.

SOLUTION: A subset $U' \subseteq U$ such that $\sum_{u \in U'} s(u) \leq B$.

MEASURE: Total weight of the chosen elements, i.e. $\sum_{u \in U'} v(u)$.

Let us construct an instance of the *B-COVER* problem as follows:

- Create a set V containing $|U|$ processors.
- Create a bijective function $f : V \longmapsto U$.
- $\forall P_i \in V,\ w_i = \frac{1}{v(f(P_i))}$.
- $\forall P_i \in V,\ f_i = s(f(P_i))$.
- $\forall P_i \in V,\ t_i = 0$.
- $E = \emptyset$ and $J_m = V$.

The graph of the *B-COVER* instance is edgeless ($E = \emptyset$), meaning that no tasks can be communicated among processors. Consequently, tasks can only be computed at the location where they are generated. A solution of the *B-COVER* instance consists in determining a subset $V' \subseteq V$ such that $\sum_{P_i \in V'} f_i \leq B$ in order to maximize the platform throughput, i.e. $\sum_{P_i \in V'} \frac{1}{w_i}$. Thus, a solution of the *B-COVER* problem instance provides a solution of the *MK* one. This proves that *B-COVER* is at least as difficult as *MK*. Since *MK* is NP-hard [15] and since the transformation is done in polynomial time, *B-COVER* is also NP-hard. $\square$

5 Heuristic Based on LP-Relaxation

LP-relaxations, i.e. relaxing the integer constraints of an integer linear program, have considerable interest since they provide the basis both for various heuristics and for the determination of bounds for the most successful integer linear programs [11]. If we replace equation (6) by the following equation: $\forall i \in J_m,\ 0 \leq x_i \leq 1$, we obtain a linear program in rational numbers, that can be solved in polynomial time. Solving the relaxed linear program associated to *B-COVER*, will give the upper bound of the optimal platform throughput reachable without exceeding the budget constraint. This bound will be our benchmark for testing and comparing our heuristic. It is important to notice that (i) this bound might not be achievable by any discrete solutions and (ii) this bound might not be tight.

Our heuristic for solving the *B-COVER* problem consists in solving the relaxed linear program in the first place, and then select in a greedy fashion the vertices P_i that have the highest g_i values without exceeding the budget constraint. We then obtain a set $M \subseteq J_m$ of master locations, and we replace equation (6) by the following equations: $\forall i \in M,\ x_i = 1$ and $\forall i \in J_m \backslash M,\ x_i = 0$. We then deal with a new linear program, equivalent to the one proposed in [14], which can be solved in polynomial time. The solution of the linear program gives rational values for all the variables, which means that we obtain a description of the resource activities during one time unit. The way to construct a schedule (i.e. where an integer number of input/output files are communicated, as well as an integer number of tasks is executed) out of the former description is shown in [14]. In this paper, our main focus is to determine optimal master locations. Reconstructing the associated schedules is therefore not relevant in our context.

6 Simulations

To verify our theoretical finding, a software simulator has been developed [9]. The inputs of the simulator are the number of vertices and edges in the graph and a probability function for the communication and the computation costs. Because input and output operations are symmetric, we let $\beta_O = 0$, i.e. tasks do not produce output files. Consequently, the masters initially hold input files, and only distributed them to the workers. We also have enforced that $\forall i, \forall j \in n(i),\ c_{i,j} = c_{j,i}$, i.e. communication times are equal between two neighbor processors, and we fixed $\gamma_{i,j} = c_{i,j}$ and $\beta_I = 1$. We hence retrieve the model proposed by Banino et al. in [14]. For the sake of generality, we let $J_m = V$, i.e. any vertex can be chosen as a master. We let the fixed cost f_i to establish master locations be equal to 1, and the per task cost t_i for operating a master be equal to 0. In other words, there are no differences in term of cost among the different potential master locations, and the influent factor for choosing master locations becomes the platform heterogeneity. We hence retrieve the model proposed by Shao et al. in [5]. As a consequence, the objective of the *B-COVER* problem becomes to maximize the platform throughput when using at most B masters. The problem becomes then similar to the *Maximal Covering Location Problem* [12] also known to be NP-hard.

The aim of these arbitrary decisions is (i) to keep the number of parameters as low as possible, while maintaining the problem complexity and (ii) to follow previous models proposed in the literature. Nevertheless, we expect our heuristic to perform better in presence of more heterogeneity, e.g. with different communicating times for $c_{i,j}$ and $c'_{i,j}$, as well as heterogeneous cost distributions, since the problem will become more specific.

Several random connected graph based on these parameters have been generated. For each graph, the throughput of our heuristic is computed, and compared to the upper bound. The following results are averaged values on 1000 random graph, each composed of 20 vertices and 39 edges. In the following simulation depicted in Fig. 2, the probability functions for the integer computation costs follow a uniform distribution on the interval $[200, 400]$, while the integer communication costs follow a uniform distribution on (i) the interval $[100, 200]$: The average computation to communication ratio (C/C) is then equal to 2, (ii) the interval $[40, 80]$: C/C is then equal to 5, and (iii) on the interval $[20, 40]$: C/C is then equal to 10.

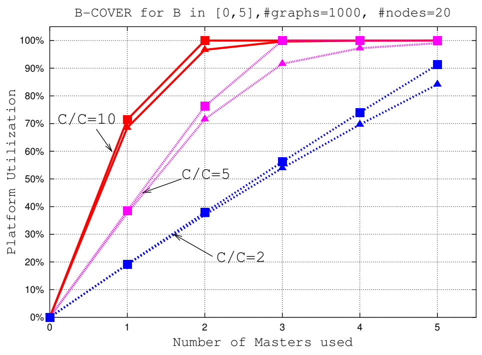


Fig. 2. Average platform utilization for the *B-COVER* problem. Square curves correspond to the upper bound, while triangle curves correspond to our heuristic

An important choice for the decision maker is deciding what level of expenditure (i.e. how many masters should be used) can be justified by the resultant throughput [12]. In our experiments for instance, 6, 3, and 2 masters would be appropriate when the C/C are equal to 2, 5 and 10 respectively.

7 Conclusion and Future Work

The problem of allocating a large number of independent, equal-sized tasks to a Grid composed of a heterogeneous collection of computing resources was considered. This paper suggests a simple but mandatory departure from traditional implementations: The need to deploy several masters to efficiently utilize currently emerging large-scale platforms. A cost model for establishing and operating master locations was provided, and the problem became to find the most profitable locations for the masters in order to optimize the platform throughput without violating a budget constraint.

On one hand, we derived pessimistic theoretical results, by showing that the *B-COVER* problem is NP-hard. On the other hand, we proposed a simple heuristic that achieves very good performance on a wide range of simulations. This heuristic can be embedded within a multi-round dynamic approach that could cope with, and respond to variations in computation speeds or network bandwidths. Similarly to the hierarchical Master-Worker approach [8], before each round a supervisor could decide how many masters should be used, where to place them and how many tasks each master will produce for the current round.

We also showed that the *B-COVER* problem presents tight connections with well-known problems from the Operation Research field. *Facility Location* and *Supply Chain Management* problems have been the subject of a wealth of research, and we believe that models and solutions to these problems can be adapted for Grid computing. Of particular interest are facility location models under uncertainty (i.e. under dynamic conditions), and facility location models with facility failures [13]. Resource failures and variations in resource availability are very likely to occur on these new emerging large-scale platforms, especially when the overall processing time is large.

This work can be extended in the following directions. First, we need approximate solutions of the *B-COVER* problem. Although our simple heuristic achieves very good performance on a wide range of simulations, there are no guarantees on how far the heuristic is from the optimal solution. Second, the *B-COVER* problem might be easier to solve (i.e. in polynomial time) on tree-shape platforms, as this is the case for the *Maximal Covering Location Problem* [16]. Finally, enabling the use of several masters transparently is a challenging task, but mandatory in order to promote the distributed computing technology. The work of Chen [6] is a step in this direction.

Acknowledgments

The author would like to thank his PhD advisor Anne C. Elster for her comments and suggestions, which greatly improved the final version of the paper. This research was supported by the Department of Computer and Information Science (IDI), at the Norwegian University of Science and Technology (NTNU).

References

1. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: Seti@home: An Experiment in Public-Resource Computing. Communications of the ACM **45** (2002) 56–61

2. Sittig, D.F., Foulser, D., Carriero, N., McCorkle, G., Miller, P.L.: A Parallel Computing Approach to Genetic Sequence Comparison: The Master Worker Paradigm with Interworker Communication. Computers and Biomedical Research **24** (1991) 152–169
3. Chien, A., Calder, B., Elbert, S., Bhatia, K.: Entropia: Architecture and Performance of an Enterprise Desktop Grid System. Journal of Parallel and Distributed Computing **63** (2003) 597
4. Beaumont, O., Legrand, A., Robert, Y.: The Master-Slave Paradigm with Heterogeneous Processors. IEEE Trans. on Parallel and Distr. Sys. **14** (2003) 897–908
5. Shao, G., Berman, F., Wolski, R.: Master/slave Computing on the Grid. In: Proceedings of the 9th Heterogeneous Computing Workshop, IEEE Computer Society (2000) 3
6. Chen, I.: A Practical Approach to Fault-Tolerant Master/Worker in Condor. Master's thesis, Dept. of Computer Science, Univ. of California at San Diego (2002)
7. Lamehamedi, H., Szymanski, B.: Data Replication Strategies in Grid Environments. In: Proc. Of the Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02) (2002)
8. Aida, K., Natsume, W., Futakata, Y.: Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm. In: Proceedings of the 3rd International Symposium on Cluster Computing and the Grid, Tokyo, Japan (2003)
9. Banino, C.: Optimizing Locationing of Multiple Masters for Master-Worker Grid Applications: A Thorough Study. Tech. Report 09/04, Dept. of Computer and Info. Science, NTNU (2004) URL: http://www.idi.ntnu.no/$\sim$banino.
10. Current, J., Daskin, M., Schilling, D.: Discrete Network Location Models. In Drezner, Z., Hamacher, H., eds.: Facility Location Theory: Applications and Methods. Springer-Verlag, Berlin (2002)
11. Krarup, J., Pruzan, P.: The Simple Plant Location Problem: Survey and Synthesis. European Journal of Operations Research **12** (1983) 36–81
12. Church, R.L., ReVelle, C.S.: The Maximal Covering Location Problem. Papers of the Regional Science Association **32** (1974) 101–118
13. Daskin, M. S., Snyder, L.V., Berter, R.T.: Facility Location in Supply Chain Design. In Langevin, A., Riopel, D., eds.: Logistics Systems: Design and Optimization. Kluwer (2005) (forthcoming).
14. Banino, C., Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Robert, Y.: Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms. IEEE Trans. Parallel Distributed Systems **15** (2004) 319–330
15. Ausiello, G., Protasi, M., Marchetti-Spaccamela, A., Gambosi, G., Crescenzi, P., Kann, V.: Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties. Springer-Verlag New York, Inc. (1999)
16. Megiddo, N., Zemel, E., Hakimi, S.L.: The Maximum Coverage Location Problem. SIAM Journal on Algebraic and Discrete Methods **4** (1983) 253–261

An OGSA-Based Bank Service for Grid Accounting Systems*

Erik Elmroth[1], Peter Gardfjäll[1], Olle Mulmo[2], and Thomas Sandholm[2]

[1] Dept. of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden
{elmroth,peterg}@cs.umu.se

[2] Dept. of Numerical Analysis and Computer Science and PDC, Royal Institute of Technology
SE-100 44 Stockholm, Sweden
{mulmo,sandholm}@pdc.kth.se

Abstract. This contribution presents the design and implementation of a bank service, constituting a key component in a recently developed Grid accounting system. The Grid accounting system maintains a Grid-wide view of the resources consumed by members of a virtual organization (VO). The bank is designed as an online service, managing the accounts of VO projects. Each service request is transparently intercepted by the accounting system, which acquires a reservation on a portion of the project's bank account prior to servicing the request. Upon service completion, the account is charged for the consumed resources. We present the overall bank design and technical details of its major components, as well as some illustrative examples of relevant service interactions. The system, which has been implemented using the Globus Toolkit, is based on state-of-the-art Web and Grid services technology and complies with the Open Grid Services Architecture (OGSA).

Keywords: Grid accounting, allocation enforcement, OGSA, SweGrid.

1 Introduction

This contribution presents the design and implementation of a bank service, constituting a key component in the SweGrid Accounting System (SGAS) [10] – a recently developed Grid accounting system, initially targeted for use in SweGrid [15]. A Grid accounting system maintains a Grid-wide view of the resources consumed by members of a *virtual organization* (VO) [6]. The information gathered by the system can serve several useful purposes, such as to allow enforcement of project quotas.

The bank is designed as an online service, handling accounts that contain the resource allocations of Grid projects. Each service request (job submission) is transparently intercepted by the accounting system, which acquires a reservation on a portion of the project account prior to job execution (cf. credit card reservations). Upon job completion, the project account is charged for the consumed resources and the reservation is released. The decentralized nature of Grids, assuming the absence of any central point of control, adds complexity to the problem. The problem is further complicated by the distributed

* This work was funded by The Swedish Research Council (VR) under contracts 343-2003-953 and 343-2003-954 and The Faculty of Science and Engineering, Umeå Univ.

resource administration in Grids, i.e., the requirement that resource owners at all times retain local control over their resources.

The rest of this paper is outlined as follows. Section 2 gives a brief introduction to the accounting issues addressed, the SweGrid environment, the concept of Grid services, and an overview of the accounting system in which the bank is a key component. The bank design is described in Section 3, including a presentation of the services constituting the bank and their relationships. Some implementation details, including a fine-grained and customizable authorization framework, are summarized in Section 4. Finally, Section 5 concludes the paper.

2 Background

The accounting system is primarily targeted towards allocation enforcement in SweGrid [15], although it has been designed to allow simple integration into different Grid environments. SweGrid is a Swedish national Grid, initially including 6 geographically distributed Linux clusters with a total of 600 CPUs dedicated for Grid usage 24 hours a day.

The computer time of SweGrid is controlled by the Swedish National Allocations Committee (SNAC) [12], which issues computer time, measured in *node hours* per month, to research projects. Node hours are assigned to projects based on their scientific merits and resource requirements, after a peer-reviewed application process. The accounting system must provide coordinated enforcement of these quotas across all sites. That is, the whole allocation may be consumed at one cluster or in parts at any number of clusters.

The design considerations for the bank service is closely related to those of the accounting system as a whole. The design is based on the assumption that a user can be a member of several VOs and participate in one or more projects within each VO. Each project's resource allocation is kept in an account, which is handled by the bank.

The information maintained by the accounting system can, e.g., form the basis for direct economic compensation, usage quota enforcement, tracking of user jobs, resource usage evaluation, and dynamic priority assignment of user requests based on previous resource usage.

The system performs soft real-time allocation enforcement. The enforcement is real-time in that resources can deny access at the time of job submission, e.g., if the allocation has been used up, and soft in that the level of enforcement strictness is subject to local policies.

2.1 Accounting System Overview

Figure 1 presents the main entities and interactions of SGAS. Entity interactions are illustrated in a scenario where a job is submitted to a computer cluster, although it could be generalized to cover a generic service request to an arbitrary Grid resource.

Each VO has an associated *Bank* service to manage the resource allocations of the VO research projects. The Bank is primarily responsible for maintaining a consistent view of the resources consumed by each research project, and enables coordinated

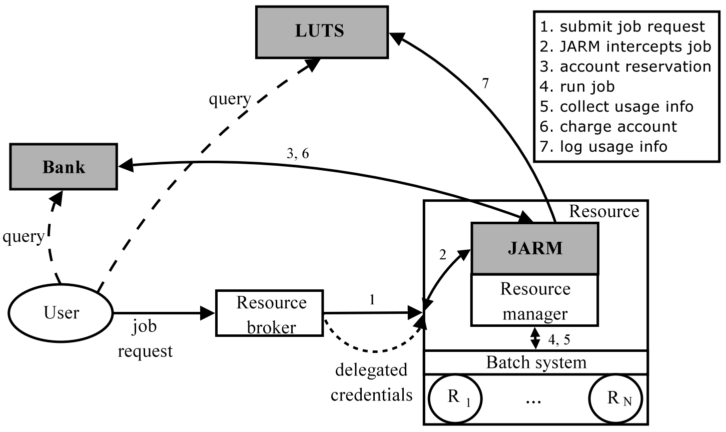


Fig. 1. Interactions among accounting system entities (shaded) during job submission

quota enforcement across the Grid sites. From a scalability perspective, a single bank per VO might seem restrictive. However, it should be stressed that the Bank service is not confined to a single site. It could be implemented as a virtual resource, composed of several distributed services, to achieve load-balancing and scalability. The *Job Account Reservation Manager* (JARM) is the (single) point of integration between SGAS and the underlying Grid environment. On each Grid resource, a JARM intercepts incoming service requests, performs account reservations prior to resource usage and charges the requester's account after resource usage. A *Log and Usage Tracking Service* (LUTS) collects and publishes Usage Records [8], holding detailed information about the resources consumed by particular service interactions.

In a typical scenario, the user, or an entity acting on behalf of the user, such as a broker, sends a job request to a resource that has been selected to execute the job. During the job request process, mutual authentication is performed and the user's credentials are delegated to the resource (1). The job request, which includes the identity of the project account, is intercepted by the resource's JARM (2), which contacts the VO Bank to acquire a time-limited reservation on a portion of the project allocation (3). Such an account reservation is referred to as a *hold*. If a hold can be granted, the JARM forwards the job request to a local resource manager (4) that runs the job and gathers information about the resources consumed by the job. At job completion, the JARM collects the usage information (5), charges the project account utilizing the hold (6), and records the usage information in a Usage Record which is logged in a LUTS (7). Any residual amount of the hold is released. Notably, a user can query both the Bank and the LUTS, e.g., for various account and job information.

2.2 The Grid Service Concept

The implementation of the accounting system, including its bank component, is based on the concept of *Grid services* as defined by the Open Grid Services Architecture (OGSA) [5]. OGSA extends the concept of *Web services* [17] by introducing a class of transient (bounded lifetime) and stateful (maintain state between invocations) Web services. Web services represent an XML-based distributed computing technology that is independent of platform, operating system and programming language. Web services are ideal for loosely coupled systems where interoperability is a primary concern.

The transient nature of Grid services makes them suitable for representing not only physical resources but also more lightweight entities/activities, such as a video conference session, a data transfer, or in this case, different parts of a Grid accounting system.

All services expose *service data*, a dynamic set of XML-encapsulated information about service metadata and local service state. Grid services provide operations for querying (service introspection) and updating service data.

The Open Grid Services Infrastructure (OGSI) [16] defines a core set of composable interfaces which are used for constructing Grid services. The bank component presented in Section 3 makes use of the OGSI interfaces for, e.g., service creation, lifetime management, and service introspection. The interfaces are defined in Web Services Description Language (WSDL) [18] portTypes and specify the operations as well as the service data exposed by each service.

3 Bank Design

The Bank component of SGAS has been designed with generality and flexibility in mind. Specifically, the bank does not assume any particular type of resources, and as such it can be integrated into any Grid environment. For example, all bank transactions are performed using *Grid credits* – an abstract, unit-less currency that can be used to charge for arbitrary resource usage. Prior to charging an account, a resource may apply any transformation function to map different kinds of resource usage into Grid credits.

Since SweGrid resources are currently homogeneous and node-hours is the only resource type being accounted for, the resource-to-Grid credits mapping is trivial. In the general case of a heterogeneous Grid environment, different types of resource usage as well as differing resource characteristics need to be considered. For example, storage utilization could be accounted for as well, and faster processors might be more expensive. In case dedicated allocations need to be maintained for different types of resources, separate accounts may be provided for each resource type.

The bank is also neutral with respect to policies. Policy decisions are left to users, resource managers and allocation authorities. The bank provides the flexibility to accommodate such policies, as well as means of enforcing them. For example, policies dictated by the allocation authority or the resource might allow a job to run even though the project allocation has been used up. However, if project quotas are not strictly enforced by a resource, the user can still decide only to perform safe withdrawals that do not exceed the project allocation.

The SGAS bank is composed of three tightly integrated Grid services (`Bank`, `Account`, `Hold`), whose relationships are illustrated in Figure 2.

Bank Service. The `Bank` service is responsible for creating as well as locating `Accounts`. The `Bank` service implements the factory pattern as provided by OGSI.

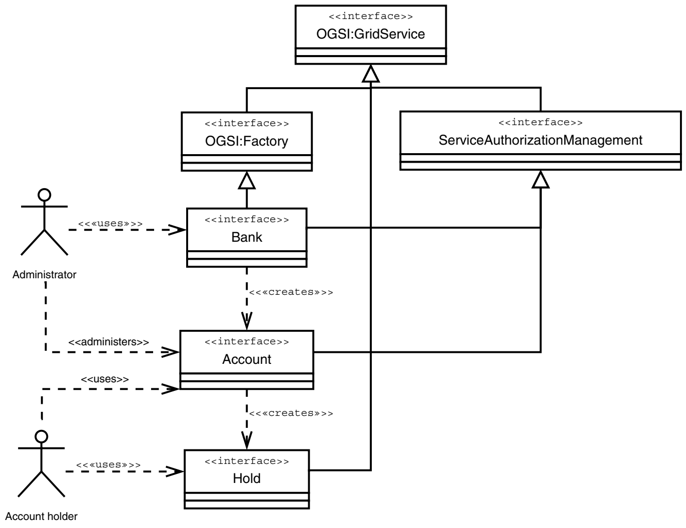


Fig. 2. Bank interface relationships

This allows the `Bank` to create new `Account` service instances. Clients can also query the `Bank` to obtain a list of the `Accounts` they are authorized to use.

Account Service. An `Account` service manages the resource allocation of a research project. A project member can request a hold on the `Account`, effectively reserving a portion of the project allocation for a bounded period of time. A successful hold request results in the creation of a `Hold` service, acting as a lock on the reserved account quota. We refer to the `Account` service that created a `Hold` service as the *parent account* of that hold. `Account` services publish transaction history and account state, which can be queried by authorized `Account` members.

The set of authorized `Bank` and `Account` users, as well as their individual access permissions, is dynamic and can be modified at run-time by setting an authorization policy, defined using XML. To this end, the `Bank` and `Account` interfaces extend the ServiceAuthorizationManagement (SAM) [10] service interface. SAM allows authorization policies to be associated with a service. The authorization policy could, e.g., contain an access control list associating a set of authorized `Account` members with their individual privileges. SAM is customizable, in that it allows different back-end authorization engines, also referred to as Policy Decision Points (PDP), to be configured with the service. Note that there is nothing `Bank`-specific about SAM; it can be used by any service requiring fine-grained management of authorization policies.

Hold Service. A `Hold` service represents a time-limited reservation on a portion of the parent account's allocation. `Holds` are usually acquired prior to job submission and committed after job completion to charge their parent account for the resources consumed by the job. `Hold` services are created with an initial lifetime and can be destroyed either explicitly or through expired lifetime. Hold expiry is controlled using the lifetime management facilities provided by OGSI. On commit or destruction, the Hold service is destroyed and any residual amount is returned to the parent account. In the case of destruction, the entire reservation is released. On commit, a specified portion of the Hold, corresponding to the actual resource usage, is withdrawn from the parent account, and a transaction entry is recorded in the transaction log.

Service Interfaces. The operations exposed by the bank services are presented in Table 1. Note that the service interfaces extend OGSI interfaces, which are used for such purposes as lifetime management, service creation and service introspection. The Bank and Account services provide batch commit operations (commitHolds), which allow resources to perform asynchronous commits of sets of Hold services in a single service invocation. The benefit of this approach is twofold, the perceived resource response time decreases, allowing higher job throughput during periods of high load, and the bank request load is reduced, improving overall system scalability. Table 2 gives an overview of the service data exposed by the bank services. Note that the service data set of each service also contains the service data of extended OGSI interfaces. Furthermore, note that since Bank and Account extend the SAM interface, these services also expose the servicePolicy service data element. Only authorized service clients are allowed to query the service data.

Table 1. The operations exposed by the SGAS bank interfaces

portType	Operation	Description
SAM	setPolicy	Set an authorization policy to be associated with service.
Bank	getAccounts	Get all accounts that caller is authorized to use.
	commitHolds	Batch commit of several Holds (created by any account in Bank).
Account	requestHold	Creates an account Hold if enough funds are available.
	addAllocation	Add a (potentially negative) amount to the account allocation.
	commitHolds	Batch commit of several Holds (created by this account).
Hold	commit	Withdraws a specified amount of the hold from the parent account.

Service Interactions. The Bank allows privileged users to create new Account services. During the lifetime of an Account, its set of members and their access rights can be modified by associating a new authorization policy with the Account, through the setPolicy operation. The project's resource allocation can be updated by an administrator by issuing a call to the addAllocation operation of the Account.

Table 2. The service data exposed by the SGAS bank interfaces

portType	**serviceData**	**Description**
SAM	servicePolicy	The authorization policy associated with the service.
Bank	none	-
`Account`	accountData	The account state: total allocation, reserved funds and spent funds.
	transactionLog	Publishes all transaction log entries.
`Hold`	holdData	The reserved amount and the identity of the parent account.

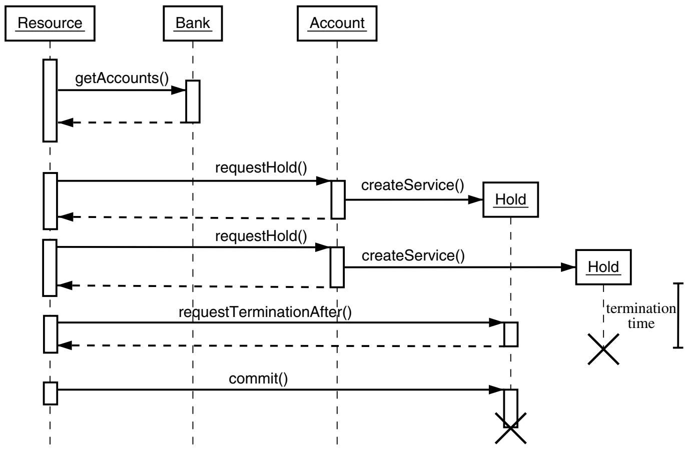


Fig. 3. `Hold` creation and usage

Figure 3 illustrates typical interactions between a resource and an `Account` service. A job is submitted by a user to a resource, which invokes the getAccounts operation on the `Bank` to get the user `Account` (unless it is specified in the job request). The resource requests a `Hold` on the `Account`, prior to job execution. The request includes an initial `Hold` expiry time. The expiry time can be reset at any time using the lifetime management operations provided by OGSI. A resource may choose to extend the `Hold` lifetime, e.g., due to a long batch queue. The commit operation is invoked by the resource after job completion to charge the `Hold`'s parent account for the consumed resources. If the `Hold` is destroyed, either explicitly or through expired lifetime, the `Hold` amount is returned to the parent `Account`.

4 Bank Implementation

Interoperability is a primary concern in Grid computing. Thus, rather than implementing our own middleware with ad-hoc protocols and communication primitives, we leverage the latest Grid standardization efforts and toolkits. Current Grid standardization focuses on Web service technologies in general and OGSA in particular.

Globus Toolkit. Our implementation is based on the Globus Toolkit (GT) [11] open-source Java reference implementation of the OGSI specification, implemented on top of the Axis SOAP engine [19]. GT provides a container framework to host Grid services and a set of tools and implementations of core OGSI interfaces, allowing developers to build and deploy custom Grid services.

By basing our solution on GT we conform to the latest standard specifications, thereby achieving desirable interoperability. Furthermore, composing our solution of toolkit primitives cuts down development time.

GT also provides a security framework orthogonal to application code, which includes mutual authentication, message encryption and message signing. These security primitives are based on the WS-SecureConversation [9], XML-Signature [3] and XML-Encryption [7] standards.

Service Implementations. All bank services are implemented using operation providers, which offer a delegation-based implementation approach where services are defined in terms of operation providers, each implementing part of the service functionality. Operation providers facilitate implementation reuse as well as a development model based on composition of primitives. For example, a service can be given the capabilities of the SAM interface simply by configuring the SAM operation provider with the service.

In order to guarantee recoverability, all services are checkpointed to a back-end database that runs embedded in the GT container. Our implementation uses Xindice [2], an open-source native XML database that conforms to the standards developed by the XML:DB group [21].

In the event of a server crash the state of all services needs to be recovered from secondary storage on server restart. The recovery is performed by means of GT service loaders. Besides enabling state checkpointing and recovery, the database solution further allows users to pose non-trivial queries against bank information, such as the transaction log, by exposing database content as service data, and embedding database query expressions in service data queries. The GT query evaluator framework allows us to redirect those service data queries to the back-end database, effectively exposing a database view through the service data framework. For example, an `Account` member can run XPath [20] queries against the transaction log to obtain specific transaction information. The XPath query approach further avoids the inconvenience of defining a query language by means of different interface operations.

Authorization Framework. The GT framework provides a declarative security infrastructure through the use of *security deployment descriptors*. Authentication method as well as handling of delegated credentials can be specified on a per-operation basis. The available framework for authorization, on the other hand, only allows authorization to be specified on a per-service basis.

As the current all-or-nothing access to a service is too coarse for our purposes, we have developed a fine-grained authorization framework [10]. Through the SAM interface, an authorization policy and an authorization engine (a PDP) can be associated with a service. The SAM design is highly flexible and customizable since it allows different authorization back-ends, using different policy languages, to be configured with a service. The current implementation uses a default authorization back-end based on the eXtensible Access Control Markup Language (XACML) [1]. Specifically, Sun's XACML PDP implementation [14] is used as the underlying authorization engine. However, successful experiments have been carried out using other authorization engines as well. Using XACML we have also included an overdraft policy, allowing an administrator to set an upper limit on acceptable account overdraft.

GT uses different handlers that intercept a SOAP [13] request before reaching the target service. To incorporate our authorization framework we provide an authorization handler that delegates the authorization decision to the target service's PDP, if one is available.

The authorization framework is orthogonal to the service implementation. That is, the service implementation is not affected by customization or replacement of the security implementation.

5 Concluding Remarks

We have presented the design and implementation of a bank service for use in a recently developed Grid accounting system. The bank, as well as the accounting system as a whole, is designed to be general and customizable, allowing non-intrusive integration into different Grid environments. The system, which is standards-based and leverages state-of-the-art Web and Grid services technology, offers user-transparent enforcement of project allocations while providing fine-grained end-to-end security.

Our planned future work includes a transition towards the Web Services Resource Framework (WSRF) [4]. Other areas that are subject to further investigation include more flexible handling of project allocations and measures of improving scalability.

Acknowledgments

We acknowledge Lennart Johnsson, Royal Institute of Technology and Bo Kågström, Umeå University, for fruitful discussions and constructive comments. We would also like to thank Martin Folkman, Uppsala University, for his work on developing an SGAS administration tool. We are also grateful for the constructive comments in the feedback from the refereeing process.

References

1. A. Anderson, A. Nadalin, B. Parducci, D. Engovatov, H. Lockhart, M. Kudo, P. Humenn, S. Godik, S. Anderson, S. Crocker, and T. Moses. eXtensible Access Control Markup Language (XACML) Version 1.0, OASIS, 2003.
2. Apache Xindice, 2004. http://xml.apache.org/xindice/.
3. M. Bartel, J. Boyer, B. Fox, B. LaMacchia, and E. Simon. XML-Signature Syntax and Processing, W3C, 2002.

4. K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe. The WS-Resource Framework: Version 1.0, 2004. http://www.globus.org/wsrf/specs/ws-wsrf.pdf.
5. I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. 2002. http://www.globus.org/research/papers/ogsa.pdf.
6. I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3):200 – 222, 2001.
7. T. Imamura, B. Dillaway, and E. Simon. XML Encryption Syntax and Processing, W3C, 2002.
8. S. Jackson and R. Lepro. Usage Record – XML Format, Global Grid Forum, 2003.
9. G. Della-Libera, B. Dixon, P. Garg, and S. Hada. Web Services Secure Conversation (WS-SecureConversation), Microsoft, IBM, VeriSign, RSA Security, 2002.
10. T. Sandholm, P. Gardfjäll, E. Elmroth, L. Johnsson, and O. Mulmo. An OGSA-Based Accounting System for Allocation Enforcement Across HPC Centers. Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04), ACM, New York, USA, November 15-19, 2004 (to appear).
11. T. Sandholm and J. Gawor. Globus Toolkit 3: A Grid Service Container Framework, 2003. http://www-unix.globus.org/toolkit/3.0/ogsa/docs/gt3_core.pdf.
12. SNAC - Swedish National Allocations Committee, 2004. http://www.snac.vr.se/.
13. SOAP Specifications, 2004. http://www.w3.org/TR/soap/.
14. Sun's XACML Implementation, Sun Microsystems, 2004. http://sunxacml.sourceforge.net/.
15. SweGrid, 2004. http://www.swegrid.se/.
16. S. Tuecke, K. Czajkowski, J. Frey, S. Graham, C. Kesselman, T. Maquire, T. Sandholm, D. Snelling, and P. Vanderbilt. Open Grid Services Infrastructure: Version 1.0, Global Grid Forum, 2003.
17. Web Services, 2004. http://www.w3.org/2002/ws/.
18. Web Services Description Language (WSDL), 2004. http://www.w3.org/TR/wsdl.
19. WebServices – AXIS, 2004. http://ws.apache.org/axis/.
20. XML Path Language (XPath), 2004. http://www.w3.org/TR/xpath.
21. XML:DB Initiative, 2004. http://xmldb-org.sourceforge.net/.

A Grid Resource Broker Supporting Advance Reservations and Benchmark-Based Resource Selection*

Erik Elmroth and Johan Tordsson

Dept. of Computing Science and HPC2N, Umeå University, SE-901 87 Umeå, Sweden
{elmroth,tordsson}@cs.umu.se

Abstract. This contribution presents algorithms, methods, and software for a Grid resource manager, responsible for resource brokering and scheduling in early production Grids. The broker selects computing resources based on actual job requirements and a number of criteria identifying the available resources, with the aim to minimize the total time to delivery for the individual application. The total time to delivery includes the time for program execution, batch queue waiting, input/output data transfer, and executable staging. Main features of the resource manager include advance reservations, resource selection based on computer benchmark results and network performance predictions, and a basic adaptation facility.

Keywords: Resource broker, scheduling, production Grid, benchmark-based resource selection, advance reservations, adaptation, Globus toolkit.

1 Introduction

The task of a Grid resource broker and scheduler is to dynamically identify and characterize the available resources, and to select and allocate the most appropriate resources for a given job. The resources are typically heterogeneous, locally administered, and accessible under different local access policies. The broker operates without global control, and its decisions are entirely based on the information provided by individual resources and information services with aggregated resource information. In addition to information about what resources are available, each resource may provide static information about architecture type, memory configuration, CPU clock frequency, operating system, local scheduling system, various policy issues, etc, and dynamic information such as current load and batch queue status. For more information on typical requirements for a good resource broker, see [3].

A reservation capability is vital for enabling co-allocation of resources in highly utilized Grids. This feature also provides a guaranteed alternative to predicting batch queue waiting times. For general discussions about resource reservations (and co-allocations), see, e.g., [5,8]. A reservation feature naturally depends on the reservation support provided by local schedulers [12] and the use of advance reservations also have implications on the utilization of each local resource [21].

* This work was funded by The Swedish Research Council (VR) under contract 343-2003-953 and The Faculty of Science and Engineering, Umeå University.

The performance differences between Grid resources and the fact that their relative performance characteristics may vary for different types of applications makes resource selection difficult, see, e.g., [9,16,19,20]. Our approach to handle this is to use a benchmark-based procedure for resource selection. Based on the user's identification of relevant benchmarks and an estimated execution time on some specified resource, the broker estimates the execution time for all resources of interest. This requires that a relevant set of benchmark results are available from the resources' information systems.

As most Grid resources still are available without performance and queue time guarantees, the adaptation feature provided by our resource broker is useful. It allows a user to request that the broker after an initial job submission strives to re-direct the job to another resource, likely to give a shorter total time to delivery.

A resource broker and scheduler is fundamental in any large-scale Grid-environment for scientific applications. Our software is mainly targeted for the NorduGrid and SweGrid infrastructures. These are both Globus-based production environments for 24 hour per day Grid usage.

The outline of the paper is as follows. Section 2 gives a brief introduction to the NorduGrid software and the general resource brokering problem. The main algorithms and techniques of our resource broker are presented in Section 3, including, e.g., support for making advance resource reservations and to select resources based on estimates of the total time to delivery. The latter feature builds on benchmark-based execution time estimates and network performance predictions. Minor extensions to the NorduGrid user interface are presented in Section 4. Sections 5 and 6 present some concluding remarks and acknowledgments, respectively, followed by a list of references.

2 Background and Motivation

Our development of resource brokering algorithms and prototype implementations are mainly focused on the infrastructure and usage scenarios typical for NorduGrid [13] and SweGrid [23]. The main Grid middleware used is the NorduGrid software [6]. The Grid resources are typically Linux-based clusters (so, in the rest of this paper, the word cluster is often used instead of the more general term Grid resource). NorduGrid includes over 40 clusters of varying size, most of them located in the Nordic countries. SweGrid currently includes six clusters, each with 100 Pentium 4 processors.

2.1 The NorduGrid Software

The NorduGrid middleware is based on standard protocols and software such as OpenLdap [14], OpenSSL [15] and Globus Toolkit version 2 [7,10]. The latter is not used in full, as some Globus components such as the GRAM (with the gatekeeper and jobmanager components) are replaced by custom made components [6].

NorduGrid makes use of the Grid Security Infrastructure (GSI) in Globus Toolkit 2. GSI specifies a public key infrastructure, and SSL (TLS) for authenticated and private communication. GSI also specifies short-term user-proxy certificates, signed by either the user or by another proxy. The proxy allows the user to access various remote resources without (re)typing passwords and to delegate authority. An extension of a subset of the Globus resource specification language (RSL) [17] is used to specify resource requirements and necessary job execution information.

The NorduGrid user interface consists of command line tools for managing jobs. Users can submit jobs, monitor the execution of their jobs and cancel jobs. The resource broker is part of the the job submission tool, `ngsub`. Other tools allows, e.g., the user to retrieve output from jobs, get a peak preview of job output and remove job output files from a remote resource. Communication with remote resources is handled by a simple GridFTP client module.

Each Grid resource runs one instance of the NorduGrid *GridFTP server*. When submitting a job, the user invokes the broker which uploads an RSL job submission request to the GridFTP server. The GridFTP server specifies plugins for custom handling of FTP protocol messages. NorduGrid makes use of these for Grid access to the local file systems of the resources, to manage Grid access control lists, and most important, for Grid job management.

Each resource also runs a *Grid manager* that manages the Grid jobs through the various phases of their execution. The Grid manager periodically searches for recently submitted jobs. For each new job, the RSL job description is analyzed, and any required input files are staged to the resource using GridFTP. The job description is translated into the language of the local scheduler, and the job is submitted to the batch system. Upon job completion, the Grid manager stages the output files to the locations specified in the job description.

2.2 The Resource Brokering Problem

In general, we can identify two major classes of Grid resource brokers, namely centralized and distributed brokers. A centralized broker manages and schedules all jobs submitted to the Grid, while a distributed broker typically handles jobs submitted by a single user only. Centralized brokers are, at least in theory, able to produce optimal schedules as they have full knowledge of the jobs and resources, but such a broker can easily become a performance bottleneck and a single point of failure. Moreover, they turn a Grid environment more into a single virtual computer than into a dynamic infrastructure for flexible resource sharing. A distributed broker architecture, on the other hand, scales well and makes the Grid more fault-tolerant, but the partial information available to each instance of the broker complicates the scheduling decisions. A hybrid type is the hierarchical broker architecture, in which distributed brokers are scheduled by centralized brokers, attempting to combine the best of two worlds. Grid systems utilizing a centralized broker include, e.g., EDG [18] and Condor [11]. Distributed brokers are implemented by, e.g., AppLeS [2] and NetSolve [4]. For further discussions on a similar classification of brokers, see, e.g., [1].

In the following we focus on algorithms and software for a distributed resource broker. Such a broker typically seeks to fulfill the user's resource requests by selecting the resources that best suit the user's application. Selecting the most suitable resources often means identifying the resources that provide the shortest Total Time to Delivery (TTD) for the job. TTD is the total time from the user's job submission to the time when the output files are stored where requested. This includes the time required for transferring input files and executable to the resource, the waiting time, e.g., in a batch queue, the actual execution time, and the time to transfer the output files to the requested location(s).

In order to manage this task, the broker needs to identify what resources are available to the user, to characterize the resources, to estimate all parts of the TTD, etc. Other requests the user may put on the broker is to make advance resource reservations, e.g., at a specific time, or to not only submit the job but also to adapt to possible changes in resource load and possibly perform job migration.

3 Resource Brokering Algorithms

Our main brokering algorithm performs a series of tasks, e.g., it processes the RSL specifications of job requests, identifies and characterizes the resources available, estimates the total time to delivery for each resource of interest, makes advance reservation of resources, performs the actual job submission. Figure 1 presents a high-level outline of the algorithm.

Input: RSL-specification(s) of one or more job requests.
Action: Select and submit jobs to the most appropriate resources.
Output: none.

```
1. Process RSL specification and create a list of all individual job requests.
2. Contact GIIS server(s) to obtain a list of available clusters.
3. Contact each resource's GRIS for static and dynamic resource information (hardware and
   software characteristics, current queue and load, etc).
4. For each job:
   (a) Select the cluster to which the job will be submitted:
         i. Filter out clusters that do not fulfill the requirements on memory, disk space, archi-
            tecture etc, and clusters that the user is not authorized to use.
        ii. Estimate TTD for each remaining resource (see Section 3.2).
            If requested, resource reservation is performed during this process.
       iii. Select the cluster with the shortest predicted TTD.
   (b) Submit the job to the selected resource.
   (c) Release any reservations made to non-selected clusters.
```

Fig. 1. Brokering algorithm overview

The input RSL specification(s) contains one or more job requests including information about the job to run (e.g., executable, input/output files, arguments), actual job requirements (e.g., amount of memory needed, architecture requirement, computer time required, requests for advance reservations), and optionally, job characteristics that can be used to make improved resource selection (e.g., listing of benchmarks with relevant performance characteristics).

In Step 1, the user's request is processed and separated into individual jobs. In Step 2, the broker identifies what resources that are available by contacting one or more Grid Index Information Services. The specific characteristics of the resources found are identified in Step 3, by contacting the Grid Resource Information Service on each individual resource. The actual brokering process is mainly performed in Step 4, where resources are evaluated, selected, and optionally reserved in advance for each job. Finally, the jobs are actually submitted and any non-utilized reservations are released. In the following presentation, we focus on the more intricate issues of performing advance reservations and on how to determine an estimate (prediction) of the total time to delivery.

Notably, this algorithm does not reorder the individual job requests (when multiple jobs are submitted in a single invocation). This can possibly be done in order to reduce the average batch queue waiting time, at least by submitting shorter jobs before longer ones given that they require the same number of CPUs. In the general case, factors such as local scheduling algorithms and competing Grid brokers make the advantage of job reordering less obvious.

3.1 Advance Resource Reservations

The advanced reservation feature makes it possible to obtain a guaranteed start time in advance. A guaranteed start time brings two advantages. It makes it possible to coordinate the job with other activities, and resource selection can be improved as the resource comparison is based on a guaranteed start time rather than on an estimate.

The reservation protocol developed supports two operations: requesting a reservation and releasing a reservation. A reservation request contains the start time and the requested length of the reservation, the requested number of CPUs, and optionally, an account to be charged for the job. Upon receiving a reservation request from the broker, the GridFTP server adds the user's local identity to the request, information unknown to the broker but required by the local scheduler. Then, the GridFTP server invokes a script to request a reservation from the local scheduler. If the scheduler accepts the request and creates the reservation, the GridFTP server sends a unique identifier and the start time of the reservation to the broker. If no reservation could be created, a message indicating failure is returned to the broker. The GridFTP server saves the reservation identifier and a copy of the user's proxy for every successful reservation, enabling subsequent identification of the user who made the reservation.

For releasing a reservation, the broker uploads a release message containing the reservation identifier and the GridFTP server confirms that the reservation is released.

Job Submission with a Reservation. Upon a successful reservation, the broker adds the received reservation identifier to the RSL job description before submitting the job to the resource.

Before the job is submitted to the local scheduler, the Grid manager analyzes the job description and detects the reservation identifier. The Grid manager inspects the saved proxy files and their associated reservation identifiers to ensure that the reservation exists. Furthermore, the Grid manager compares the proxy used to submit the job with the one used to make the reservation, ensuring that no user can submit jobs to another user's reservation by spoofing the reservation id. The job is not submitted to the local scheduler unless the specified reservation exists and was created by the user submitting the Grid job.

When the job finishes executing on the resource, the Grid manager may remove the reservation, allowing the user to run only the requested job. Alternatively, resources may allow the user to submit more jobs to the reservation once the first has finished. The configuration of the Grid manager determines the policy to be used.

The advance resource reservation feature requires a reservation capability in the local scheduler. The current implementation supports the Maui scheduler [22], although any local scheduler may be used. Support for other schedulers can easily be added by adapting the scripts interacting with the local scheduler (see, e.g., [12]).

3.2 Estimating the Total Time to Delivery

The estimation of the total time to delivery (TTD), from the user's job submission to the final delivery of output files to requested storage requires that the time to perform the following operations is estimated:

1. Stage in: transfer of input files and executable to the resource,
2. Waiting, e.g., waiting in batch queue and for operation 1 to complete,
3. Execution, and,
4. Stage out: transfer of output files to requested location.

Notably, the waiting time is here defined as the maximum of the time for stage in and all other waiting times before the job actually can start to execute. The estimated TTD is given as the sum of estimated times for operations 2, 3, and 4. Sometimes, the time for stage out cannot be estimated, due to lack of information about output file sizes. In these cases that part is simply omitted from the TTD used for comparisons. Below, we summarize how we make these estimates.

Benchmark-Based Time Predictions. The execution time estimate needs to be based both on the performance of the resource and the characteristics of the applications. This issue is made slightly more intricate by the fact that the relative performance difference between different computing resources typically varies with the character of the application. In order to circumvent this problem, we give the user the opportunity to specify one or more benchmarks with performance characteristics similar to those of the application. This information is given together with an execution time estimate on a resource with a specified benchmark result. Based on this information and benchmark results for each individual resource, we make execution time estimates for all resources of interest. In doing this, we assume linear scaling of the application in relation to the benchmark, i.e., a resource with a benchmark result a factor k better is assumed to execute the application a factor k faster.

We remark that a good execution time estimate serves two purposes. First, a sufficient but short estimated execution time may lead to an earlier job start, due to standard batch system scheduling algorithms. Second, a good estimate is more likely not to be too short, and hence reduces the risk for job preemption by the local scheduler.

The user can specify k benchmarks as triples $\{b_i, r_i, t_i\}, i = 1, \ldots, k$, where b_i is the benchmark name, r_i is the benchmark result on a system where the application requires the time t_i. The broker matches these specifications with the benchmark results provided by each cluster. For each requested benchmark that is available at the resource, an execution time for the application is predicted.

If the cluster does not provide a result for a requested benchmark, the corresponding time estimate is taken to be a penalty factor c times the longest execution time estimated from other benchmarks for that cluster. The penalty factor can be configured by the user. As a default value, $c = 1.25$ is used.

When comparing the performance of different clusters during the resource selection procedure, one part of the TTD estimate for each resource is given from this procedure. The value used is the average execution time estimate of the k estimates obtained from different benchmarks. At job submission, after the selection procedure, an execution time must be included in the request sent to the resource. In order to reduce the risk for scheduler preemption, this execution time is chosen as the maximum of the k estimates calculated.

Network Performance Predictions. The time estimation for the stage in and stage out procedures are based on the actual (known) sizes of input files and the executable file, user-provided estimates for the sizes of the output files, and network bandwidth predictions.

The network bandwidth predictions are performed using the Network Weather Service (NWS) [24]. NWS combines periodic bandwidth measurements with statistical methods to make short-term predictions about the available bandwidth.

3.3 Job Queue Adaptation

Information gathered about the state of a Grid is necessarily old. Network load and batch queue sizes may change rapidly, new resources may appear and others become unavailable. The load predictions used by the broker as a basis for resource selection will become out-of-date. Nevertheless, more recent information will always be available as Grid resources periodically advertise their state. With this in mind, the broker can keep searching for better resources once the initial job submission is done. If a new resource that is likely to result in a earlier job completion time is found (taking into account all components of TTD, including file restaging), the broker migrates the job to the new resource. This procedure is repeated until the job starts to execute on the currently selected resource.

4 User Interface Extensions

We have extended the standard NorduGrid user interface with some new options and added some new attributes to the NorduGrid RSL, making the new features available to users.

Benchmarks and Advance Resource Reservations. In order to make use of the feature of benchmark-based execution time prediction, the user must provide relevant benchmark information as described by the following example. Assume that the user knows that the performance of the application my_app is well characterized by the NAS benchmarks LU, BT and CG. For each of these benchmarks, the user needs to specify a benchmark result and an expected execution time on a system corresponding to that benchmark result. Notably, the expected execution time must be given for each benchmark, as the benchmark results may be from different reference computers. This is specified using the new RSL attribute `benchmarks`.

Figure 2 illustrates how the user specifies that the application requires 65 minutes on a cluster where the results for the NAS LU and BT benchmarks class C are 250 and 200, respectively. The estimated execution time is 50 minutes on an (apparently different) cluster where the CG benchmark result is 90.

```
&(executable = my_app)(stdin = my_app.in)
 (stdout = my_app.out)(benchmarks = (nas-lu-c 250:65)
 (nas-bt-c 200:65)(nas-cg-c 90:50))
```

Fig. 2. Sample RSL request including benchmark-based execution time predictions

Network Transfers. In the example in Figure 3, the job involves transfer of large input and output files. The broker will determine the actual sizes of the input files when estimating the transfer time for these. The new, optional, RSL attribute `outputfilessizes` allows the user to provide an estimate of the size of the job output. As shown in the figure, the user need not include all output files in the `outputfilessizes` relation. Estimated file sizes can specified in bytes or with any of the suffixes kB, MB or GB. The NorduGrid middleware uses the `inputfiles` and `outputfiles` attributes as a replacement for `file_stage_in` and `file_stage_out` from the RSL specification.

```
&(executable = my_program)(arguments = params input)
                                (stdout = logfile)
 (inputfiles = (params gsiftp://host1/file1)
                 (input http://host2/file2))
 (outputfiles = (results gsiftp://host3/my_program.results)
                (data gsiftp://host3/my_program.data)
                (logfile gsiftp://host4/my_program.log))
 (outputfilessizes = (results 230MB)
                     (data 5GB))
```

Fig. 3. Sample RSL request with specifications required to estimate file transfer times

Command Line Options. In addition to the RSL extension, the broker supports some new command line options. The option `-A` is used to request the broker to perform queue adaptation. The reservation feature is invoked using the option `-R`. The option `-S` is used to build a pipeline between jobs, so that output from one job is used as input to the next.

5 Concluding Remarks

The resource broker presented is developed with focus on the NorduGrid software and the NorduGrid and SweGrid production environments. It includes support for making advance resource reservations and selects resources based on benchmark-based execution time estimates and network performance predictions. The broker is a built-in component of the user's submission software, and is hence a user-owned broker acting with no need for global control, entirely basing its decisions on the dynamic information provided by the resources.

Acknowledgments

We acknowledge Åke Sandgren, High Performance Computing Center North (HPC2N), Umeå, Sweden, for technical support. We are also grateful for the constructive comments in the feedback from the refereeing process.

References

1. C. Anglano, T. Ferrari, F. Giacomini, F. Prelz, and M. Sgaravatto. WP01 report on current technology. http://server11.infn.it/workload-grid/docs/DataGrid-01-TED-0102-1_0.pdf.
2. F. Berman and R. Wolski. The AppLeS project: A status report. In N. Koike, editor, *Proceedings of the 8th NEC Research Symposium*, 1997.
3. J. Brooke and D. Fellows. Draft discussion document for GPA-WG – Abstraction of functions for resource brokers. http://grid.lbl.gov/GPA/GGF7_rbdraft.pdf.
4. H. Casanova and J. Dongarra. Netsolve: A network server for solving computational science problems. *Int. J. Supercomput. Appl.*, 11(3):212–223, 1997.
5. K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems. In Feitelson D.G. et al., editors, *Job Scheduling Strategies for Parallel Processing*, volume 2537 of *Lecture Notes in Computer Science*, pages 153–183, Berlin, 2002. Springer-Verlag.
6. P. Eerola, B. Kónya, O. Smirnova, T. Ekelöf, M. Ellert, J.R. Hansen, J.L. Nielsen, A. Wäänänen, A. Konstantinov, J. Herrala, M. Tuisku, T. Myklebust, F. Ould-Saada, and B. Vinter. The NorduGrid production Grid infrastructure, status and plans. In *Proc. 4th International Workshop on Grid Computing*, pages 158–165. IEEE CS Press, 2003.
7. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Int. J. Supercomput. Appl.*, 11(2):115–128, 1997.
8. I. Foster, C. Kesselman, C. Lee, B. Lindell, K. Nahrstedt, and A. Roy. A distributed resource management architecture that supports advance reservations and co-allocation. In *7th International Workshop on Quality of Service*, pages 27–36, Washington - Brussels - Tokyo, 1999. IEEE.
9. I. Foster, J.M. Schopf, and L. Yang. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *Proceedings of the ACM/IEEE SC2003: International Conference for High Performance Computing and Communications*, 2003.
10. Globus. http://www.globus.org.
11. M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, June 1988.
12. J. MacLaren. Advance reservations state of the art. http://www.fz-juelich.de/zam/RD/coop/ggf/graap/sched-graap-2.0.html.
13. NorduGrid. http://www.nordugrid.org.
14. OpenLDAP. http://www.openldap.org.

15. OpenSSL. http://www.openssl.org.
16. K. Ranganathan and I. Foster. Simulation studies of computation and data scheduling algorithms for data Grids. *Journal of Grid Computing*, 1(1):53–62, 2003.
17. The Globus Resource Specification Language RSL v1.0. http://www-fp.globus.org/gram/rsl_spec1.html.
18. M. Ruda, C. Anglano, S. Barale, L. Gaido, A.Guarise, S. Lusso, A. Werbrouck, S. Beco, F. Pacini, A. Terracina, A. Maraschini, S. Cavalieri, S. Monforte, F. Donno, A, Ghiselli, F. Giacomini, E. Ronchieri, D. Kouril, A. Krenek, L. Matyska, M. Mulac, J. Popisil, Z. Salvet, J. Sitera, J.Visek, M. Vocu, M. Mezzadri, F. Prelz, M. Sgaravatto, and M. Verlato. Integrating Grid tools to build a computing resource broker: activities of datagrid WP1. In *CHEP'01 computing in high energy and nuclear physics*.
19. W. Smith, I. Foster, and V. Taylor. Predicting application run times using historical information. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, Berlin, 1999. Springer-Verlag.
20. W. Smith, I. Foster, and V. Taylor. Using run-time predictions to estimate queue wait times and improve scheduler performance. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 202–219, Berlin, 1999. Springer-Verlag.
21. W. Smith, I. Foster, and V. Taylor. Scheduling with advanced reservations. In *14th International Parallel and Distributed Processing Symposium*, pages 127–132, Washington - Brussels - Tokyo, 2000. IEEE.
22. Supercluster.org. Center for HPC Cluster Resource Management. http://www.supercluster.org.
23. Swegrid. http://www.swegrid.se.
24. Rich Wolski. Dynamically forecasting network performance using the network weather service. *Journal of Cluster computing*, 1(1):119–132, 1998.

The Dragon Graph: A New Interconnection Network for High Speed Computing

Jywe-Fei Fang

St.John's and St.Mary's Institute of Technology, Taipei 251, Taiwan, R.O.C.

Abstract. A new interconnection network called the Dragon graph, is proposed. A Dragon graph is a variation of the hypercube and the cube-connected-cycle with constant degree four. The Dragon graph gains many advantages. It has a smaller diameter and cost than the comparable cube-connected-cycle. It is node-symmetric and edge-symmetric. A routing algorithm and a broadcasting algorithm are proposed in this paper.

1 Introduction

The aim of studying interconnection networks and their combinatorial properties is to find good topologies for massively parallel computing. Ideally, attributes of a good topology include the *diameter*, *cost*, *communication* and *symmetry* [4]. The diameter is the maximum distance of each pair of nodes in a graph. So, we can use it to measure the maximum communication delay. The degree can be used to measure the hardware complexity of each processor. Usually, an interconnection network with a small degree implies potentially a large diameter. It appears that there is a tradeoff between the degrees and diameters of interconnection networks. Thus, there is a commonly used metric, called cost, which is the product of diameter and degree.

It is widely recognized that interprocessor communication is one of the most important issues for interconnection networks because the communication problem is the key issue to many parallel algorithms [1]. *Routing* and *broadcasting* are two primitive communication problems. The former is to send a message from a source to a destination node and the latter is to distribute the same message from a source node to all other nodes. They appear in applications such as matrix operations (e.g., matrix multiplication, factorization, inversion, transposition), database operation (e.g. polling, master-slave operation) and transitive closure algorithms [4].

An interconnection network is *node-symmetric* if and only if for any two nodes U and V, there exists an automorphism of the network that maps U to V. An interconnection network is *edge-symmetric* if and only if for any two edges e and g, there exists an automorphism of the network that maps e to g. An interconnection network with *node-symmetry* and *edge-symmetry* implies that it can be implemented regularly.

In recent years, among many interconnection networks, the *hypercube* has been the focus of many researchers due to its structural regularity, potential for the parallel computation of various algorithms, and the high degree of fault tolerance capabilities [8]. However, the hypercube has a practical limitation. Each processor of the n-dimensional hypercube is connected to n other processors. Consequently, the hypercube is unfeasible for practical implementation of massively computing because its large fanout logarithmically proportional to the number of processors. To overcome the disadvantage of the hypercube, some variations of hypercube with constant degree such as *cube-connected cycles*, *butterfly networks*, *debruijn networks*, *shuffle exchange graphs* have been proposed [4].

The cube-connected cycle has received considerable attention [7]. Researchers have devoted themselves to various issues of cube-connected cycles. Meliksetian and Chen devised an optimal routing algorithm for the cube-connected cycle and derived the the exact diameter [6]. Kuo and Fuchs proposed two reconfigurable architectures of cube-connected cycles, which are capable of tolerating classes of multiple failures [3]. Chen and Lau presented a new layout of the cube-connected cycle which uses less than half the area of the Preparata-Vuillemin layout [2,7]. On the other hand, various variations of the cube-connected cycle have been proposed. Lo and Chen proposed a generalized cube-connected cycle called *k-ccc* to enhance the extensibility [5]. Sebastian et al. proposed the folded cube-connected cycle, another variation of the cube-connected cycles, to improve the diameter. However, the both topologies are neither node-symmetric nor edge-symmetric. Thus, to lay out them is potentially complex. In this paper, the *Dragon graph*, a new variation of the cube-connected cycle, with constant degree(degree four) is proposed. It gains many advantages. For example, It is both node-symmetric and edge-symmetric. Thus, it is potential to be implemented regularly. We derive the topological properties such as diameter and cost of the Dragon graph. To compare with the corresponding cube-connected cycle, it has a smaller diameter and cost. We also propose a routing algorithm and a broadcasting algorithm for the Dragon graph.

The rest of this paper is organized as follows. In Section 2, the structure of the Dragon graph is described. We also present some notations and background that will be used throughout this paper. In Section 3, we propose a routing algorithm for the Dragon graph. In Section 4, a broadcasting scheme of the Dragon graph is introduced. Some topological properties of the Dragon graph are discussed in Section 5. Finally, in Section 6, we give some concluding remarks.

2 Notation and Backgrounds

Let $H(n)$ denote an n-dimensional hypercube comprising of 2^n nodes. $H(n)$ is constructed by numbering the nodes from 0 to $2^n - 1$ and linking each pair of nodes whose addresses differ by exactly one bit position. A link is assigned a link number i if it connects two nodes whose addresses are different at exactly the ith bit position (starting with the least significant bit as bit 0).

The n-dimensional cube-connected-cycle denoted by $CCC(n)$ is derived from the n-dimensional binary hypercube by replacing each node of the hypercube with a cycle of n nodes. Each node of $CCC(n)$ is labelled by a pair (w, i), where w is the label of the node in the hypercube which corresponds to the cycle and i is the position of the node within the cycle. Two nodes (w, i) and (w', i') of $CCC(n)$ are connected by an edge if and only if either

(1) $w = w'$ and $i \equiv (i+1)$ mod n.

(2) $i = i'$ and w differs from w' by exactly the ith bit.

An edge of the first type(second type) is called *cycle*(*hypercube*) *edge*. It is easy to see that $CCC(n)$ is not *edge-symmetric* because cycle edges and hypercube edges play different roles in the network. In Figure. 1, the structure of $CCC(3)$ is displayed.

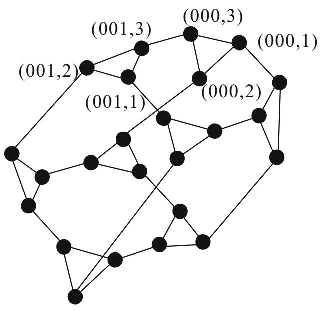


Fig. 1. The structure of $CCC(3)$

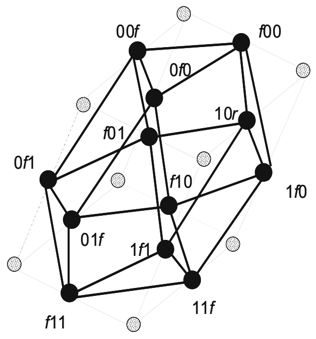


Fig. 2. The structure of $DG(3)$, where the dash line and dash node show the hypercube that is the $DG(3)$ derived from

The n-dimensional Dragon graph denoted by $DG(n)$ is derived from the $CCC(n)$ by merging each pair of nodes which is connected with each other by the hypercube edge in $CCC(n)$. Each node U of $DG(n)$ is labelled by a string of n symbols $u_{n-1}u_{n-2}\ldots u_i\ldots u_1u_0$ over set $0, 1, f$, where f is to label the hypercube edge in the corresponding $CCC(n)$. For example, a pair of nodes (000,3) and (001,3) of $CCC(3)$ will be merged to a node $00f$ of $DG(3)$. It is easy to see that f exists exactly once in each node address of $DG(n)$. In Figure. 2, the structure of $DG(3)$ is shown..

Definition 1. Let $U = u_{n-1}u_{n-2}\ldots u_i\ldots u_1u_0$, where u_i is f, be a node in $DG(n)$. The flag bit of U, denoted by $flag(U)$, is defined as i .

Two nodes of the $DG(n)$, U with flag bit i and V with flag bit j are connected by an edge if and only if both

(1) $j \equiv i+1 \bmod n$ or $j \equiv i-1 \bmod n$, and

(2) each bit except the ith bit and the jth bit of U and V is common.

For example, $0f0$ is connected to $f00$, $f10$, $00f$ and $01f$. $f00$ is connected to $00f$, $10f$, $0f0$ and $1f0$. In fact, each node of Dragon graph is connected to four neighbors, (r,1) neighbor,(r,0) neighbor, (l,0) neighbor and (l,1) neighbor.

Definition 2. The $(r,1)((r,0))$ neighbor of a node $U = u_{n-1}u_{n-2}\dots u_i \dots u_1u_0$ with flag bit i can be derived by shifting the flag bit to bit $i-1$ mod n(that is, $u_{i-1\ mod\ n}$ is replaced by symbol f)and complementing u_i with symbol 1(0).

Definition 3. The $(l,1)$,$((l,0))$ neighbor of a node $U = u_{n-1}u_{n-2}\dots u_i \dots u_1u_0$ with flag bit i can be derived by shifting the flag bit to bit $i+1$ mod n(that is, $u_{(i+1)\ mod\ n}$ is replaced by symbol f)and complementing u_i with symbol 1(0).

For example, $(r,1)$ neighbor,$(r,0)$ neighbor, $(l,1)$ neighbor and $(l,0)$ neighbor of $0100f$ is $f1001$, $f1000$, $010f1$ and $010f0$.

Definition 4. The binary sequence of node $U = u_{n-1}u_{n-2}\dots u_1u_0$ with flag bit i, denoted by $Binseq(U)$,is defined as $u_{n-1}u_{n-2}\dots u_{i+1}u_{i-1}\dots u_1u_0$.

For example, $Binseq(10f1)$ is 101. If nodes are with the same binary sequence, They are called in the common binary sequence class. For Example, $f010$, $0f10$, $01f0$, $010f$ are in the common binary sequence class 010.

Definition 5. In a common binary sequence class, compare the nodes by their flag bit position, the node with higher(lower) flag bit position, is called the senior(junior).

For example, $f010$ and $0f10$ are the seniors of $01f0$, and $010f$ is the junior of $01f0$.

Definition 6. Specifically, $U = u_{n-1}u_{n-2}\dots u_1u_0$ with flag bit i is called as the ith element of the common binary sequence class $u_{n-1}u_{n-2}\dots u_{i+1}u_{i-1}\dots u_1u_0$.

By the structure of a Dragon graph and the above definition, we have the following proposition.

Proposition 1. In a common binary sequence class, the ith element is connected to the $(i-1)$th element.

For example, in the common binary sequence 1101 of $DG(5)$, each pair neighbors of $f1101$, $1f101$, $11f01$, $110f1$, $1101f$ are connected.

Definition 7. Let $U = u_{n-1}u_{n-2}\dots u_1u_0$ and $V = v_{n-1}v_{n-2}\dots v_1v_0$ be two nodes in $DG(n)$. The ith bit is referred as an $active\ bit$ if and only if both of u_i and v_i are not flag bit and $u_i \neq v_i$.

3 Routing Algorithm

Given a source node and a destination node of an interconnection network. The routing problem is to find a path such that a message can be transmitted from the source node to the destination node. To present the idea of this algorithm, suppose that source node and destination node are $f0000$ and $1101f$. First,if there exists any active bit, it is required to "eliminate" these active bits, that is, to find a path from source node to an intermediate node with no active bit between this node and destination node. For example, shifting the flag bit rightward and "eliminate" the active bit, a path can be derived as $f0000$, $1f000$, $11f00$, $110f0$. Likewise, shifting the flag bit leftward and "eliminate" the active bit, a path can be derived as $f0000$, $1000f$, $100f0$, $10f10$, $1f010$. There is a choice between rightward and leftward. The shorter one is selected. Second, it is required to shift the flag bit to the "right" position, that is, to find a path from the intermediate node to the destination node. For example, from the intermediate node $110f0$, shifting the flag bit rightward, a path can be derived as $110f0$ and $1101f$. Likewise, shifting the flag bit leftward, a path can be derived as $110f0$, $11f10$, $1f010$, $f1010$ and $1101f$. There is also a choice between rightward and leftward. The shorter one is selected, too. Clearly, this routing algorithm can rout a message from source node to destination node.

We propose the routing algorithm formally as follows. Suppose that the routing message is included by the address of destination node $D = d_{n-1}d_{n-2}\dots d_1d_0$.

Algorithm 1. Routing algorithm of $DG(n)$

Step 1. When a node $V = v_{n-1}v_{n-2}\dots v_1v_0$ with flag bit i, receives the message included by the node address of destination $D = d_{n-1}d_{n-2}\ d_1d_0$ with the flag bit j,

Step 1, starting with the bit i, to find the nearest active bits between V and D rightward and leftward, respectively.

performs by the following conditions:

1) : If there exists any active bit and $i \neq j$,

if the rightward is nearer than the leftward, rout the message to the (l, d_i) neighbor. Otherwise, rout the message to the (r, d_i) neighbor.

2) : If there exists any active bit and $i = j$,

if the rightward is nearer than the leftward, rout the message to the (l, 0) neighbor. Otherwise, rout the message to the (r, 0) neighbor.

3) : If there exists no active bit and and $i \neq j$,

if ($j > i$ and $j - i < [n/2]$) or ($j < i$ and $i - j \geq [n/2]$), rout the message to the (l, d_i) neighbor. Otherwise, rout the message to the (r, d_i) neighbor.

4) : If there exists no active bit and and $i = j$, then $V = D$, terminates the route.

4 Broadcasting Algorithm

Broadcasting on an interconnection network can be realized based on a broadcasting tree that is a spanning tree rooted at the source node. In this section, we introduce a broadcasting algorithm for Dragon graphs by building a broadcasting tree. Since $DG(n)$ is node-symmetric, the same broadcasting algorithm can be applied to each node of $DG(n)$. Without loss of generality, suppose that the source node is $r00\dots0$. This algorithm comprises two phases.

Phase 1, to build a primitive tree.

Initialization: The broadcasting tree has one node $r00\dots0$ as root.

While the flag bit of a leaf node is not 0, appending its (r,0) neighbor and (r,1) neighbor as left child and right child, respectively. In Figure 3, the primitive tree of $DG(4)$ is shown.

To build the primitive tree is very simple. However, the tree does not include each node of the whole Dragon graph. In the Phase 2, the remainder will be appended. Observe that each node and its left child are in the same binary sequence. Thus, the juniors of a node will be its descendants in the primitive tree. Recall that according to the Proposition 1, in a common binary sequence class, the ith element is connected to the $(i-1)$th element.

Phase 2, appending phase. If a node is right child, append its seniors in a linear array one by one to the primitive tree.

In Figure 4, the broadcasting tree of $DG(4)$ is shown.

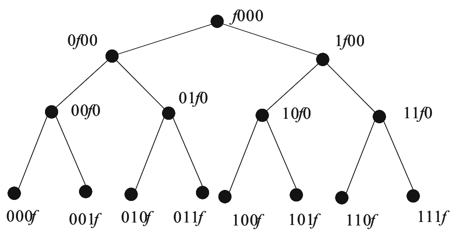


Fig. 3. The primitive tree of $DG(4)$

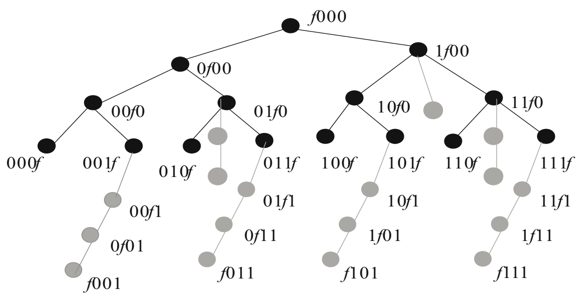


Fig. 4. The broadcasting tree of $DG(4)$

Table 1. Symmetric properties of some constant degree hypercubic networks

Networks	Node symmetry	Edge symmetry
Shuffle exchange	None	None
Debruijn network	None	None
Butterfly network	None	None
CCC	Yes	None
Dragon graph	Yes	Yes

5 Properties of Dragon Graphs

Since $CCC(n)$ is node-symmetric, clearly, $DG(n)$ is node-symmetric, too. Furthermore, because only the cycle edges of $CCC(n)$ are reserved in $DG(n)$, $DG(n)$ is also edge-symmetric. In Table 1, we show the symmetric properties of some constant degree hypercubic networks [2]. Clearly, The Dragon graph is superior to the other interconnection networks in symmetric properties.

The distance between two nodes of an interconnection network is defined as the length of their shortest path. The diameter of an interconnection network is defined as the maximal distance between each pair of nodes.

By using $Algorithm$ 1, each node can route a message to any other node within $n-1+(n-1)/2$ hops ($n-1$ hops for "eliminating" the active bit and $(n-2)/2$

Table 2. The degrees, diameters and costs of $DG(n)$ and $CCC(n)$

$DG(n)$				$CCC(n)$			
n	degree	diameter	cost	n	degree	diameter	cost
3	4	3	12	3	3	7	21
4	4	4	16	4	3	9	27
5	4	5	20	5	3	12	36
6	4	7	28	6	3	14	42
7	4	8	32	7	3	17	51
8	4	10	40	8	3	19	57
9	4	11	44	9	3	22	66
10	4	13	52	10	3	24	72
11	4	14	56	11	3	27	81
12	4	16	64	12	3	29	87
13	4	17	68	13	3	32	96
14	4	19	76	14	3	34	102
15	4	20	80	15	3	37	111
16	4	22	88	16	3	39	117
17	4	23	92	17	3	42	126
18	4	25	100	18	3	44	132
19	4	26	104	19	3	47	141
20	4	28	112	20	3	49	147

hops for shifting the flag bit to the "right" position) in $DG(n)$ for n is an even number. Likewise, each node can route a message to any other node within $n-1+(n-3)/2$ hops ($n-1$ hops for "eliminating" the active bit and $(n-3)/2$ hops for shifting the flag bit to the "right" position) in $DG(n)$ for n is an odd number. In Table 2, we summarize the degrees, diameters and costs of an $DG(n)$ and an $CCC(n)$. The diameters and costs of a $DG(n)$ are smaller than the comparable $CCC(n)$.

6 Conclusions

In this paper, the Dragon graph, a new variation of the hypercube and the cube-connected-cycle with constant degree(four) has been proposed. The Dragon graph gains many advantages. It is with a smaller diameter and cost than the comparable cube-connected-cycles. It is node-symmetric and edge- symmetric. In this paper, we also proposed a routing algorithm which is not complex, thus it can be applied to practical implementation. By building a broadcasting tree, we have presented a broadcasting algorithm which is also potential to be implemented.

Acknowledgements

The author would like to thank reviews for their valuable comments and suggestions. This work was supported in part by the National Science Council of the Republic of China under the contract number: NSC92-2213-E-129-010.

References

1. S. G. Akl,*The Design and Analysis of Parallel Algorithms,* Prentice-Hall, 1989.
2. G. Chen and F. C. M. Lau, "Tighter layouts of the cube-connected cycles," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, pp. 182-191, 2000.
3. S. Y. Kuo and W. K. Fuchs, "Reconfigurable cube-connected cycles architectures," *Journal of Parallel and Distributed Computing,* vol. 9, pp. 1-10, 1990.
4. F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays,Trees, Hypercubes,* California, Mogran Kaufmann, 1992.
5. H. Y. Lo and J. D. Chen, "A routing algorithm and generalization for cube-connected cycles," *IEICE Trans. Information Systems,* vol. E80-D, pp. 829-836, 1997.
6. D. S. Meliksetian and C. Y. R. Chen, "Optimal routing algorithm and the diameter of the cube-connected cycles," *IEEE Trans. Parallel and Distributed Systems,* vol. 4, pp. 1172-1178, 1993.
7. F. P. Preparata and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computations," *Commun. ACM,* vol. 25, pp. 300-309, 1981.
8. Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Trans. Computers,* vol. 37, pp. 867-872, 1988.
9. M. P. Sebastian, P. S. N. Rao and L. Jenkins, "Properties and performance of folded cube-connected cycles," *Journal of Systems Architecture,* vol. 44, pp. 359-374, 1998.

Speeding up Parallel Graph Coloring

Assefaw H. Gebremedhin[1,⋆], Fredrik Manne[2], and Tom Woods[2]

[1] Computer Science Dept., Old Dominion University
Norfolk, VA 23529-0162, USA
assefaw@cs.odu.edu

[2] University of Bergen, N-5020 Bergen, Norway
{Fredrik.Manne,tomw}@ii.uib.no

Abstract. This paper presents new efficient parallel algorithms for finding approximate solutions to graph coloring problems. We consider an existing shared memory parallel graph coloring algorithm and suggest several enhancements both in terms of ordering the vertices so as to minimize cache misses, and performing vertex-to-processor assignments based on graph partitioning instead of random allocation.

We report experimental results that demonstrate the performance of our algorithms on an IBM Regatta supercomputer when up to 12 processors are used. Our implementations use OpenMP for parallelization and Metis for graph partitioning. The experiments show that we get up to a 70 % reduction in runtime compared to the previous algorithm.

1 Introduction

A graph coloring asks for an assignment of as few colors (or positive integers) as possible to the vertices V of a graph $G = (V, E)$ so that no two adjacent vertices receive the same color. This is often a crucial stage in the development of efficient, parallel algorithms for many scientific and engineering applications, see [8] and the references therein for examples. As a graph coloring is often needed to perform some later task concurrently, it is natural to consider performing the coloring itself in parallel. In the current paper, we consider how to parallelize fast and greedy coloring algorithms where vertices are colored one at a time and each vertex is assigned the smallest possible color.

Several papers have appeared in the literature over the last couple of years dealing with this issue [5,7,8,11]. We consider the algorithm in [8] and show how its performance can be enhanced both by paying attention to its sequential memory access pattern, and by employing graph partitioning to assign vertices to processors.

The rest of the paper is organized as follows: In Section 2 we present relevant background information on parallel graph coloring, in Section 3 we motivate our methods, in Section 4 we report on experimental results, before we conclude in Section 5.

2 Parallel Graph Coloring

The problem of finding a legal coloring using the minimum number of colors is known to be NP-hard for general graphs [6]. Moreover, it is a difficult problem to approximate [3].

⋆ Supported by the U.S. National Science Foundation grant ACI 0203722

However, in practice greedy sequential coloring heuristics have been found to be quite effective [2].

Previous efforts in designing fast efficient parallel graph coloring algorithms have been centered around various ways of computing an independent set in parallel. This research direction was initiated by Luby's algorithm for computing a maximal independent set in parallel [10,15]. The first algorithms proposed along this line enforce that one makes exactly one choice for the color of any vertex (i.e. no backtracking). To achieve this, while coloring a vertex v, the colors of already colored neighbors of v must be known, and none of the uncolored neighbors of v can be colored at the same time as v. The work of Jones and Plassmann [12], Gjertsen et al. [9], and Allwright et al. [1] follow this approach.

All of these algorithms are based on the distributed memory model where the vertices of a graph are partitioned into the same number of components as there are processors. Each component including information about its inter- and intra-processor edges is assigned to and colored by one processor.

To overcome the restriction that two adjacent vertices on different processors cannot be colored at the same time, Johansson [11] proposed a parallel algorithm where each processor is assigned exactly one vertex. The vertices are then colored simultaneously by randomly choosing a color from the interval $[1, \Delta + 1]$, where Δ is the maximum vertex degree in the graph. This may lead to an inconsistent coloring, and hence one may have to repeat the process recursively for the vertices that did not obtain legal colors. This algorithm was further investigated by Finocchi et al. [5] who performed extensive sequential simulations both of this algorithm and a variant where the upper-bound on the range of legal colors is initially set to be smaller than $\Delta + 1$ and then increases only when needed.

Gebremedhin and Manne [8] developed a parallel graph coloring algorithm suitable for shared memory computers. Here it is assumed that the number of processors p is substantially less than the number of vertices n and each processor is assigned n/p vertices. Each processor colors its vertices sequentially, at each step assigning a vertex the smallest color not used by any of its neighbors (both on and off-processor). An inconsistent coloring arises only when a pair of adjacent vertices that reside on different processors are colored simultaneously. Inconsistencies are then detected in a subsequent phase and resolved in a final sequential phase. Figure 1 outlines this scheme as presented in [8]. This algorithm was later extended to various coloring problems in numerical optimization including *distance-2* ($d2$) coloring, a case in which a vertex v is required to receive a color distinct from the colors of vertices within distance 2 edges from v [7]. This shared memory approach is the only algorithm that we know of that has been shown through parallel implementations to actually give lower running time as more processors are applied. In the current work we look at various ways in which this algorithm can be enhanced to run even faster.

In [8] it is shown that the expected number of conflicts in most cases is small and the resulting running time is bounded by $O(\Delta n/p)$. Although Phase 0 states that the vertices be randomly distributed among the processors, practical experiments showed that the best running time was obtained by keeping the original ordering of the vertices of the graph.

```
**GM-Algorithm**($G = (V, E)$)
*Phase 0 : Partition*
    Randomly partition $V$ into $p$ equal blocks $V_1, \dots, V_p$. Processor $P_i$ is
    responsible for coloring the vertices in block $V_i$.
*Phase 1 : Pseudo-color*
    for $i = 1$ to $p$ do in parallel
        for each $v \in V_i$ do
            assign a legal color to $v$, paying attention
            to already colored vertices (both on and off-processor).
*Phase 2 : Detect conflicts*
    for $i = 1$ to $p$ do in parallel
        for each $v \in V_i$ do
            if $\exists (v, w) \in E$ s.t. $color(v) = color(w)$ and $v \leq w$
                $L_i \leftarrow L_i \cup \{v\}$
*Phase 3 : Resolve conflicts*
    Color the vertices in the conflict list $L = \cup L_i$ sequentially.
```

Fig. 1. The GM-algorithm as presented in [8]

3 Speeding up the Algorithm

In the following we describe the various enhancements we have considered in order to speed up the GM-algorithm. Sections 3.1 and 3.2 deal with issues related to the partitioning of the graph onto the processors and the subsequent internal ordering, while Section 3.3 deals with algorithmic issues.

In a parallel application the graph is usually already distributed among the processors in a reasonable way. This also includes distinguishing between interior and boundary vertices since the communication volume typically grows as a function of the number of boundary vertices while the amount of computation grows as a function of the total number of vertices assigned to a processor. This is true both for shared and distributed memory parallel computers. Thus it is not unreasonable to make the initial assumption that the graph is already partitioned in a reasonable way, with the vertices on each processor classified as either interior or boundary, and with an internal ordering of the vertices such that traversing the graph should be efficient.

3.1 Sequential Optimization

It is a well known fact that sequential optimization can in many cases yield speedups comparable to those obtained by parallelization. The main objective here is to make the code more cache friendly by reusing data elements already in the cache.

The GM-algorithm uses a compressed adjacency list representation of a graph. Each vertex has a pointer into this list that shows where its neighbors are located. The actual color values are stored in a separate vertex-indexed array.

When accessing the list of neighbors of a vertex, the information will be in consecutive memory locations. However, accessing the colors of neighboring vertices might lead to sporadic memory access patterns with ensuing cache misses. This is particularly true if the neighbors are scattered rather than being clustered.

In light of this, if the vertex set is randomly permuted as was suggested in Phase 0 of the GM-algorithm, one would expect the neighbors of a vertex to be randomly distributed throughout the set and thus cause a large number of cache misses. This could be one explanation why this scheme performed worse than keeping the natural order of the vertices.

To reduce the number of cache misses, one would order the vertices in such a way that the neighbors of each vertex span as few cache lines as possible. One way this can be solved approximately is by employing a bandwidth or envelope size reducing ordering on the graph such as the reverse Cuthill-McKee ordering [4]. For an adjacency matrix representation this would have the effect of clustering the non-zero elements close to the diagonal.

3.2 Graph Partitioning

There exist several graph partitioning packages that can be used for partitioning the vertices of a graph into a predefined number of components in such a way that the number of vertices in each component is nearly the same and the number of cross-edges (edges with endpoints in different components) is as small as possible. In a parallel application this can be used to partition the graph into the same number of components as there are processors and then assign one component to each processor.

Graph partitioning is also exploited in the parallel coloring algorithms presented in [12] and [9] where one takes advantage of the fact that *boundary* vertices (vertices incident on inter-processor edges) are the only vertices that call for communication, and *interior* vertices (vertices incident only on intra-processor edges) can be colored concurrently.

The use of graph-partitioning software can help lower the run-time of the GM-algorithm in three different ways.

i) Cross-processor memory accesses. In Phase 1 and 2 of the GM-algorithm, part of the run-time associated with the coloring of a vertex v can be divided into two separate parts: the time required to obtain the color of an adjacent on-processor vertex, and the time required to obtain the color of an adjacent off-processor vertex. For most architectures, the latter will be larger since it most likely involves accessing memory that is associated with another processor. For this reason, one would like to maximize the number of neighbors of v that are allocated to the same processor as v itself. Thus an assignment of the vertices to processors that keeps the number of cross-edges low will also reduce the time spent on off-processor memory accesses.

ii) Number of conflicts. In the context of the GM-algorithm, minimizing the number of cross-edges also reduces the number of conflicts since cross-edges are the only edges that can give rise to conflicts. This is because vertices in the same component will be colored (sequentially) by the same processor. Reducing the number of conflicts will subsequently reduce the time needed by Phase 3 of the algorithm. In fact, the bound of $O(\overline{\delta}p)$ on the number of expected conflicts shown in [8] can easily be improved to $O(\overline{\delta}pm'/m)$. Here, $\overline{\delta}$ is the average vertex degree, m' is the number of cross-edges, and m is the total number of edges in the graph.

iii) Vertices that need to be checked for conflicts. As we have already seen, any conflict that arises in Phase 1 of the GM-algorithm will involve at least one boundary vertex. Thus by distinguishing between interior and boundary vertices, it is possible to avoid checking the interior vertices for conflicts in Phase 2. This should save up to half the processing time spent on the interior vertices. Although reducing the number of boundary vertices is not the main objective of graph-partitioning software, we expect this number to correlate with the number of cross-edges.

Finally we note that in the case where a d2-coloring is desired, any conflict will still include at least one boundary vertex. This is because the d2-neighbors of an interior vertex v will either be in the same component as v or a boundary vertex of some adjacent component. Thus even in this case it is sufficient to check the boundary vertices to detect all possible conflicts. For distance-k coloring, $k > 2$ this is no longer true as the distance relationship then stretches further into each component.

3.3 Conflict Reduction

In [7] it was shown that for dense graphs the number of conflicts could become sufficiently high that the final sequential conflict resolution step starts to dominate the overall runtime. Also for dense graphs the effect of graph partitioning is less pronounced.

One solution for reducing the number of conflicts suggested in [11] is to assign a vertex a random color from the interval $[1, \Delta + 1]$. However, with this scheme one is very likely to end up using close to $\Delta + 1$ colors while the actual number of colors needed is much smaller.

To reduce the number of colors used, one would like each processor to choose a small legal color when coloring a vertex. If each processor always chooses the lowest possible color, the number of conflicts is likely to increase. This is particularly true during the early steps of the coloring process where there are relatively few forbidden colors. Thus one would like different processors to choose different low-numbered colors. In the following paragraphs we describe two solutions that have been suggested to this problem. Both are based on randomization.

Gebremedhin et al. [7] suggested that one use a geometrically distributed random variable to determine which of the available legal colors to use. This was implemented by processing the available legal colors in increasing order, each time choosing a color with probability q. The process terminates as soon as a successful color choice is made. The disadvantages of this method is that it requires some fine-tuning of the parameter q and for small values of q, one might need to generate a large number of random numbers for each vertex. We note that the latter shortcoming can be alleviated by an *inverse transformation* method to simulate the required random variable and hence require only one random number generation [16].

Finocchi et al. [5] recently suggested the "Trivially Hungarian" (TH) method where the color of a vertex v is initially restricted to be chosen from the range $[1, r]$, where $r = deg(v)/s$, $deg(v)$ is the degree of v, and s is an a priori determined *shrink factor* of the algorithm. Only when there are no legal colors in the given range can the bound r be increased to $1 + \min\{c, deg(v)\}$, where c is the largest color used in the neighborhood of v. Again, this algorithm requires that a suitable value of s be found. The TH-method has not previously been implemented in an actual working parallel code.

Table 1. Structural properties of graphs (left) and run-times for $d1$ and $d2$ coloring using various vertex orderings (right)

Name	$\|V\|$	$\|E\|$	*density*	Δ	$\bar{\delta}$	Natural	Random	RCM
mrng4	7,533,224	14,991,280	5.3×10^{-7}	4	3.98	3.6/17.2	10.5/73.9	2.5/10.4
auto	448,695	3,314,611	3.3×10^{-5}	37	14.77	0.5/8.0	0.8/12.1	0.4/5.0

4 Experiments

In the following we report on experiments performed on an IBM p690 Regatta supercomputer using up to 12 Power4 1.3 Ghz processors. Our objective is to show how the various strategies described in Section 3 can be used to speed up the GM-algorithm. The algorithms have been implemented in Fortran 90 and parallelized using OpenMP. Metis [14] was used for graph partitioning.

4.1 Vertex Ordering

Here we present experimental results that demonstrate the effect of vertex ordering on the practical run-time of the coloring algorithms.

Due to space limitations, we only provide results for two representative graphs from our testbed, both of which are from finite element methods [13]. The left part of Table 1 lists relevant structural properties of these graphs. The column labeled $density$ shows the quantity $\frac{2|E|}{|V|\times(|V|-1)}$. The column marked Δ gives the maximum vertex degree while column $\bar{\delta}$ gives the average vertex degree in the graph. In the right part of Table 1, the columns labeled *Natural*, *Random*, and *RCM* show the time (in seconds) used by a sequential greedy coloring algorithm when the vertices are visited in their natural order, in a random order, and in the reverse Cuthill-McKee order, respectively. For each ordering, run-times for a $d1$-coloring and a $d2$-coloring are provided.

Table 1 shows that a random vertex ordering destroys any available locality and hence increases the running time by a factor of nearly three for $d1$-coloring and by a factor of four for $d2$-coloring. The RCM ordering reduces the running time by 31% and 20% for the $d1$-coloring and by 40% and 37% for the $d2$-coloring.

4.2 Vertex to Processor Mapping

Figure 2 shows the parallel running times of our algorithms for $d1$-coloring as the number of processors is increased from 2 to 12. A substantial speedup was observed in going from one to two processors even if the parallel algorithm performs nearly twice as many operations as the sequential algorithm. (We believe this is due to access to more cache when using more than one processor.) We do not show this as it would disrupt the figures.

The line labeled *Original* refers to the basic GM-algorithm as described in Section 2 run on the graph with the vertices in their natural order and partitioned into contiguous blocks of equal size (block-partition). The line labeled *RCM* corresponds to an RCM ordering of the vertices prior to a block-partition. The line *RCM+Metis* shows the case where the vertices are ordered using RCM and the graph partitioned using Metis. Finally

RCM+Metis+Local shows when both RCM and Metis are used and the interior vertices are not checked for inconsistencies. It should be noted that Metis only specifies to which partition a particular vertex should belong, it does not alter the relative order among the vertices within a particular partition.

In Figure 2, for a fixed number of processors, the decrease in running time relative to the basic algorithm ranges from 63% to 75% with a mean of 68%. Although not reported, we have also made experiments using only Metis, which gave similar results to the ones obtained using only RCM. In all cases, the number of conflicts was observed to be consistently low and did not influence the overall running time. As expected, we also observed that for a fixed number of processors, the number of conflicts decreased with the number of boundary vertices (and cross-edges). Moreover, the numbers of colors used remains close to that used by the sequential greedy algorithm.

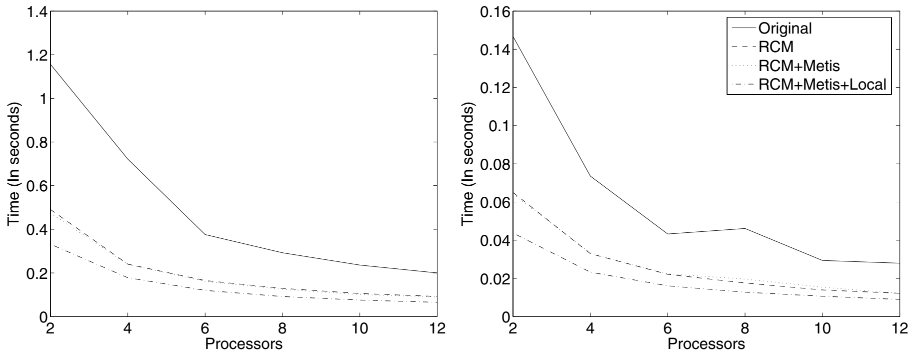


Fig. 2. $d1$-coloring of the graphs *mrng4* (left) and *auto* (right)

To help further explain the decrease in running time we also present Figure 3 which shows the percentage of boundary vertices while using the original, RCM, and Metis ordering. Applying Metis to the RCM ordered graph has little effect as Metis is fairly insensitive to the original ordering.

Note that RCM ordering improves locality but not as much as Metis does. Thus since applying either only RCM or only Metis gives approximately the same running times, the gain obtained by the Metis ordering where there are fewer cross-processor memory accesses and a smaller set of boundary vertices seems to be compensated for by the more efficient memory access pattern given by the RCM ordering.

As is evident from Figure 3 the percentage of boundary vertices increases as the number of processors increases. But at the same time the number of interior vertices handled by each processor decreases. Thus one might suspect that the obtained speedup is mainly due to each processor spending less and less time on the interior vertices. This would be similar to the algorithm by Gjertsen et al. [9]. To see that this is not the case, note that already when using four processors more than 80 % of the vertices of both graphs are on the boundary but the speedup still continues.

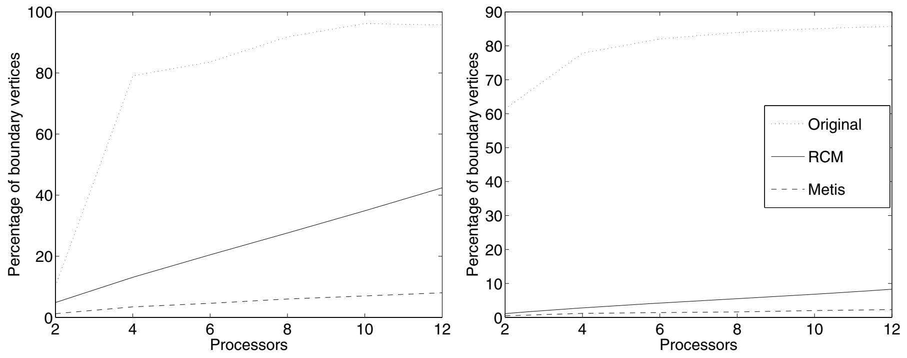


Fig. 3. Percentage of boundary vertices for graphs *auto* (left) and *mrng4* (right)

In Gjertsen et al. [9] the interior and boundary vertices are colored separately. We found that in our setting it is advantageous to store and color the interior and boundary vertices interleaved. We believe that this is due to external memory references being evened out in time and thus avoiding communication congestion.

4.3 Randomization

In the experiments in the previous section, the number of conflicts that had to be resolved was small enough that the final sequential phase did not influence the overall running time. In this section we report on experiments where this is not the case. Specifically we test how the randomized conflict reduction schemes described in Section 3.3 can be applied to improve the running time.

For these experiments we report results obtained on a random graph (*d2-random*) with 30,000 vertices, 8,999,700 edges, a maximum vertex degree of 699, and an average vertex degree of 600. This graph was chosen such that the original parallel $d2$-coloring algorithm would give a large number of conflicts. We have also performed experiments using graphs tailored toward $d1$-coloring. These gave results (omitted for space considerations) similar to those reported here.

For the randomized algorithms we performed similar experiments as those discussed in Section 4.2. However, these did not yield any significant improvements in the running times over the original algorithm but kept the results fairly stable. The reason for this is that irrespective of the chosen ordering technique, almost all the vertices were on the boundary. Hence the experiments shown here are based on the basic algorithm with the vertices in their natural order.

The main parameter that must be determined for the TH-algorithm is the shrinking factor s. In [5] a value between four and six is suggested for $d1$-coloring. Our configuration differs from theirs in that we rely on a shared memory model while they assume a distributed memory model. Also, they apply the algorithm recursively to the vertices that do not receive a legal color after the first round while our approach is to recolor illegally colored vertices in a sequential phase.

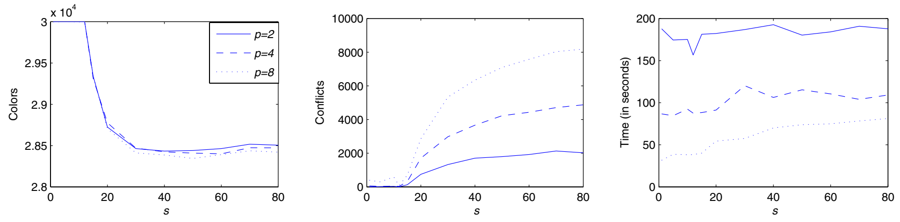


Fig. 4. The TH-algorithm run on the *d2-random* graph

Figure 4 shows the performance of the TH-algorithm for a $d2$-coloring of the graph *d2-random* using different values of s. The leftmost figure shows the number of colors used, the middle figure shows the number of conflicts, and the rightmost figure depicts the time used. As is evident from Figure 4, the shrinking factor in the TH-algorithm is critical for its performance. If it is set too small, the algorithm can use more than double the number of colors used by the sequential greedy algorithm. On the other hand, if the shrinking factor is set too large, the number of conflicts, and consequently, the running time increases. The interval for the shrinking factor that produces the best values, both in terms of the number of colors used and run-time, is fairly small. Interestingly, in our experiments we found that the best results are obtained if the shrinking factor is set in such a way that the number of initial available colors is equal or almost equal to the number of colors used by the sequential greedy algorithm. We note that for practical purposes this might be difficult to estimate a priori.

In order to compare the TH algorithm with the GMP-algorithm [7], we ran the GMP-algorithm on the *d2-random* graph using various values for the parameter q. Due to space limitations we do not show these numbers here but we observed that the optimal value of q in terms of speed seems to decrease as the number of processors increases. In terms of best possible running times, the TH and the GMP-algorithm have fairly similar values, but obtaining these requires fine tuning of the algorithms.

5 Conclusion

One open interesting question is: Given a partition, how should one order the vertices within each component to minimize cache misses both locally and between processors?

Even though the randomized methods seem to be promising in terms of handling dense graphs, more work remains to be done to determine how these need to be tailored for specific applications. It would also be of interest to know how these techniques could be applied in a distributed memory setting.

References

1. J. Allwright, R. Bordawekar, P. D. Coddington, K. Dincer, and C. Martin, *A comparison of parallel graph coloring algorithms*, NPAC technical report SCCS-666, Northeast Parallel Architectures Center at Syracuse University, 1994.
2. T. F. Coleman and J. J. More, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. Numer. Anal., 1 (1983), pp. 187–209.

3. P. CRESCENZI AND V. KANN, *A compendium of NP optimization problems*. http://www.nada.kth.se/~viggo/wwwcompendium/.
4. E. CUTHILL AND J. MCKEE, *Reducing the bandwidth of sparse symmetric matrices*, in proceedings of ACM NAT. Conf., 1969, pp. 157–172.
5. I. FINOCCHI, A. PANCONESI, AND R. SILVESTRI, *Experimental analysis of simple, distributed vertex coloring algorithms*, in Proc. 13th ACM-SIAM symposium on Discrete Algorithms (SODA 02), 2002.
6. M. R. GAREY AND D. S. JOHNSON, *Computers and Intractability*, Freeman, 1979.
7. A. H. GEBREMEDHIN, F. MANNE, AND A. POTHEN, *Parallel distance-k coloring algorithms for numerical optimization*, in proceedings of Euro-Par 2002, vol. 2400, Lecture Notes in Computer Science, Springer, 2002, pp. 912–921.
8. A. H. GEBREMEDHIN AND F. MANNE, *Scalable parallel graph coloring algorithms*, Concurrency: Practice and Experience, 12 (2000), pp. 1131–1146.
9. R. K. GJERTSEN JR., M. T. JONES, AND P. PLASSMANN, *Parallel heuristics for improved, balanced graph colorings*, J. Par. Dist. Comput., 37 (1996), pp. 171–186.
10. A. GRAMA, A. GUPTA, G. KARYPIS, AND V. KUMAR, *Introduction to Parallel Computing, 2ed*, Addison Wesley, 2003.
11. Ö. JOHANSSON, *Simple distributed $\delta + 1$-coloring of graphs*, Inf. Proc. Letters, 70 (1999), pp. 229–232.
12. M. T. JONES AND P. PLASSMAN, *A parallel graph coloring heuristic*, SIAM J. Sci. Comput., (1993), pp. 654–669.
13. G. KARYPIS. Private communications.
14. G. KARYPIS AND V. KUMAR, *Multilevel k-way partitioning scheme for irregular graphs*, J. Par. Dist. Comp., 48 (1998), pp. 96–129.
15. M. LUBY, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., (1986), pp. 1036–1053.
16. S. M. ROSS, *Introduction to Probability Models, 7ed*, Academic Press, 2000.

On the Efficient Generation of Taylor Expansions for DAE Solutions by Automatic Differentiation[⋆]

Andreas Griewank[1] and Andrea Walther[2]

[1] Department of Mathematics, Humboldt-Universität Berlin, Germany
griewank@mathematik.hu-berlin.de
[2] Institute of Scientific Computing, Technische Universität Dresden, Germany
awalther@math.tu-dresden.de

Abstract. Under certain conditions the signature method suggested by Pantiledes and Pryce facilitates the local expansion of DAE solutions by Taylor polynomials of arbitrary order. The successive calculation of Taylor coefficients involves the solution of nonlinear algebraic equations by some variant of the Gauss-Newton method. Hence, one needs to evaluate certain Jacobians and several right hand sides. Without advocating a particular solver we discuss how this information can be efficiently obtained using ADOL-C or similar automatic differentiation packages.

1 Introduction

The differential algebraic systems in question are specified by a vector function

$$F(t, \mathbf{y}) \;\equiv\; F(t, \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n) \quad \text{with}$$
$$F : \mathbb{R}^{1+m} \equiv \mathbb{R} \times \mathbb{R}^{1+m_1} \times \mathbb{R}^{1+m_2} \times \cdots \times \mathbb{R}^{1+m_n} \mapsto \mathbb{R}^n \;\; .$$

Here, t denotes the independent “time” variable, n is the dimension of the state space and the $\mathbf{y}_j$ are $(1 + m_j)$-dimensional vectors whose rth component $y_{j,r}$ represents the rth derivative of the state space component $y_j \equiv \mathbf{y}_{j,0}$. However, this relation is really of no importance as far as the pure automatic differentiation task is concerned.

In principle, F can be an arbitrary algebraic mapping from $\mathbb{R}^{1+m}$ to $\mathbb{R}^n$ but for our purposes it should be defined by an evaluation procedure in a high level computer language like Fortran or C. Then, the technique of automatic differentiation (AD) offers an opportunity to provide derivative information of any order for the given code segment by applying the chain rule systematically to statements of computer programs. For that purpose, the code is decomposed into a typically very long sequence of simple evaluations, e.g. additions, multiplications, and calls to elementary functions such as $\sin(x)$ or $\exp(x)$. The derivatives with respect to the arguments of these operations can be easily calculated. Exploiting the chain rule yields the derivatives of the whole sequence of statements with respect to the input variables. Depending on the starting point of this methodology—either at the beginning or at the end of the chain of computational steps—one distinguishes between the forward mode and the reverse mode

⋆ This work was supported by the DFG research center MATHEON, Mathematics for Key Technologies in Berlin, and DFG grant WA 1607/2-1.

of AD. Using the forward mode, one computes the required derivatives together with the function evaluation in one sweep. Applying the reverse mode of AD, the derivative values are propagated during a backward sweep. Hence after a function evaluation, one starts computing the derivatives of the dependents with respect to the last intermediate values and traverses backwards through the evaluation process until the independents are reached. A comprehensive exposition of these techniques of AD can be found in [7].

For many DAEs the computational graph related to the code segment to evaluate $F(t, \mathbf{y})$ will be of quite manageable size, but still we will try to keep the number of sweeps through it as low as possible. Moreover, we will try to amortize the overhead of each sweep by performing reasonably intense calculations for each graph vertex, which represents an intermediate quantity in the evaluation of the algebraic function $F(t, \mathbf{y})$.

The structure of the paper is the following. In Section 2 we discuss the efficient computation of first and higher-order derivatives using automatic differentiation. The structural analysis required for the signature method suggested by Pantelides [9] and Pryce [10,11] is the subject of Section 3. New drivers of the AD-tool ADOL-C [8] are presented for the first time in Section 4. They are tailored especially for the use in the DAE context. Preliminary numerical results illustrating the effort to compute higher-order derivatives are discussed in Section 5. Section 6 contains some conclusions and an outlook.

2 Computing First and Higher Order Derivatives

Throughout the paper we assume that the time t has been shifted so that its current value is simply $t = 0$. Then we obtain for any analytic path

$$\mathbf{y}(t) = \sum_{r=0}^{\bar{r}} \mathbf{y}^{(r)} t^r$$

a corresponding value path

$$F(t, \mathbf{y}(t)) = \sum_{r=0}^{\bar{r}} t^r \hat{F}_r(0, \mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(r)}) + \mathcal{O}(t^{\bar{r}+1})$$

The coefficient functions $\hat{F}_r : \mathbb{R}^{1+m*r} \mapsto \mathbb{R}^n$ are analytic provided this is true for F as we assume for simplicity. To compute the desired higher-order information, we will first examine the derivative computation for intrinsic function. For a given Taylor polynomial

$$x(t) = x_0 + x_1 t + x_2 t^2 + \cdots + x_{\bar{r}-1} t^{\bar{r}} \in \mathbb{R}^n$$

a naive implementation to compute higher-order derivatives could be based on "symbolic" differentiation. This approach would yield for a general smooth $v(t) = \varphi(x(t))$ the derivative expressions

$$\begin{aligned}
v_0 &= \varphi(x_0) \\
v_1 &= \varphi_1(x_0)\, x_1 \\
v_2 &= \varphi_2(x_0)\, x_1\, x_1 + \varphi_1(x_0)\, x_2 \\
v_3 &= \varphi_3(x_0)\, x_1\, x_1\, x_1 + 2\,\varphi_2(x_0)\, x_1\, x_2 + \varphi_1(x_0)\, x_3 \\
v_4 &= \varphi_4(x_0)\, x_1\, x_1\, x_1\, x_1 + 3\varphi_3(x_0)\, x_1\, x_1\, x_2 \\
&\quad +\varphi_2(x_0)\,(x_2\, x_2 + 2\, x_1\, x_3) + \varphi_1(x_0)\, x_4 \\
v_5 &= \quad \ldots
\end{aligned}$$

Hence, the overall complexity grows rapidly in the highest degree $\bar{r}$ of the Taylor polynomial. To avoid these prohibitively expensive calculations the standard higher-order forward sweep of automatic differentiation is based on Taylor arithmetic [2] yielding an effort that grows like $\bar{r}^2$ times the cost of evaluating $F(t, \mathbf{y})$. This is quite obvious for arithmetic operations as shown below. For a general elemental function φ, one finds also a recursion with quadratic complexity by interpreting φ as solution of a linear ODE. The following table illustrates the resulting computation of the Taylor coefficients for a simple multiplication and the exponential function:

$v(t) =$	Recurrence for $k = 1 \ldots \bar{r} - 1$	OPS	MOVES
$x(t) * y(t)$	$v_k = \sum\limits_{j=0}^{k} x_j y_{k-j}$	$\sim \bar{r}^2$	$3\,\bar{r}$
$\exp(x(t))$	$k v_k = \sum\limits_{j=1}^{k} j v_{k-j} x_j$	$\sim \bar{r}^2$	$2\,\bar{r}$

Similar formulas can be found for all intrinsic functions. This fact permit the computation of higher-order derivatives for the vector function $F(t, \mathbf{Y})$ as composition of elementary components. The AD-tool ADOL-C [8] uses the Taylor arithmetic as described above to provide an efficient calculation of higher-order derivatives. Furthermore, the AD-tools FADBAD [1] and CppAD [4] use the same approach to compute higher-order information.

In ODE and DAE solving, the coefficients $\mathbf{y}^{(r)}$ are generated in increasing order using successive values of the residuals $\hat{F}_r$. For that purpose, we have to compose the input coefficients of the vectors

$$\mathbf{y}^{(r)} = [\mathbf{y}_1^{(r)}, \mathbf{y}_2^{(r)}, \ldots, \mathbf{y}_n^{(r)}]$$

from the values $y_{j,s}$ obtained so far. This simple application of the chain rule is the only extra procedure we have to attach to our AD software to facilitate the calculation of the desired Taylor coefficients. Specifically, using ADOL-C we must set

$$y_{j,s}^{(r)} = y_{j,r+s}/s!$$

because the computations performed by ADOL-C are based on the unscaled Taylor coefficients.

If the $\hat{F}_s$ for $s \leq r$ are reevaluated from scratch every time this requires r sweeps and thus a computational effort of order r^3 times the cost of evaluating the underlying algebraic mapping $F(t, \mathbf{y})$. As observed in [7, Section 10] there are at least two ways in which this effort can be reduced to being quadratic in r. The first option is to store and retrieve the partial Taylor polynomials of all intermediate quantities that occur during the evaluation of F. This has been done in the Fortran package ATOMFT [5] for the solution of ODEs by the Taylor series method.

The second possibility is to exploit the property that F_r is linear in all $\mathbf{y}^{(s)}$ with $s > r/2$ so that in fact

$$\begin{aligned} &F_r(0, \mathbf{y}^{(0)}, \mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(r)}) \\ &= F_r(0, \mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(s-1)}, 0, \ldots 0) + \sum_{k=s}^{r} A_{k-s}(0, \mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(k-s)})\mathbf{y}^{(s)} \end{aligned}$$

Here the $A_s(0, \mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(k-s)}) \in \mathbb{R}^{n \times n}$ for $s \leq r$ are the Taylor coefficients of the Jacobian path $J(0, \mathbf{y}(t))$. They can also be evaluated by standard AD methods and are needed anyway if one wishes to compute sensitivities of the Taylor coefficients with respect to the basis point $\mathbf{y}^0$ in an implicit Taylor method. In contrast to the save and restore option, exploiting the linearity reduces the number of sweeps through the computational graph essentially to the logarithm of the maximal order $\bar{r}$.

3 Structural Analysis in Terms of the Jacobian *J*

The structural analysis used by Pantelides [9] has become part of professional simulation software [3,6] and has been applied successfully to a wide variety of systems. Nevertheless, it has to be mentioned, that Pantelides' algorithm applied to DAEs of index 1 may perform an arbitrarily high number of iterations and differentiations [12]. This behaviour is due to the fact that the structural index of the DAE may exceed the index of the DAE and that the differentiation needed by Pantelides' algorithm relate to the structural index. However, the present paper focuses on the derivative computation. Therefore, these possibly difficulties are just a side note for computing the consisted point. They might be overcome in the future by a better suited determination method for the nonnegative shifts mentioned below.

In the structural analysis used by Pantelides [9] and Pryce [10,11], the elements σ_{ij} of the signature matrix Σ are defined by

$$\sigma_{ij} = \begin{cases} \max\{r \mid y_{j,r} \text{ occurs in } F_i\} \\ -\infty \quad \text{if no component of } \mathbf{y}_j \text{ occurs in } F_i \end{cases},$$

where F_i denotes the ith component function of F. Here, "$y_{j,r}$ occurs in F_i" means that the value of the latter depends nontrivially on the former, which leads to its occurrence in a symbolic expression for F_i. The signature matrix is used to determine vectors $c = (c_i)_{i=1}^n$ and $d = (d_j)_{j=1}^n$ of nonnegative shifts. It is shown in [10] that if a solution

$$\mathbf{Y}^* = (y^*_{1,0}, \ldots, y^*_{1,d_1}, \ldots, y^*_{n,0}, \ldots, y^*_{n,d_n})$$

of the equations

$$\left.\begin{array}{c} F_1^{(0)}, F_1^{(1)}, \ldots, F_1^{(c_1)} \\ \vdots \\ F_n^{(0)}, F_n^{(1)}, \ldots, F_n^{(c_n)} \end{array}\right\} = 0$$

exists and the system Jacobian J with

$$J_{ij} = \begin{cases} \dfrac{\partial F_i}{\partial y_{j,d_j-c_i}} & \text{if } y_{j,d_j-c_i} \text{ occurs in } F_i \\ 0 & \text{otherwise} \end{cases}$$

is nonsingular at the solution, then $\mathbf{Y}^*$ is a consistent point of the DAE at time t. The required derivative information for constructing J can be obtained by a single reverse sweep in vector mode to evaluate the rectangular Jacobian

$$J(0,\mathbf{y}) \equiv [J_1(0,\mathbf{y}), J_2(0,\mathbf{y}), \ldots, J_n(0,\mathbf{y})] \in \mathbb{R}^{n\times m} \qquad \text{with}$$
$$J_j(0,\mathbf{y}) \equiv \frac{\partial F(0,\mathbf{y})}{\partial \mathbf{y}_j} \in \mathbb{R}^{n\times(1+m_j)} .$$

Irrespective of the size of m, the operations count for this will be about n times that of evaluating $F(t,\mathbf{y})$ by itself.

To compute a solution $\mathbf{Y}^*$ one has to solve a sequence of underdetermined systems of nonlinear equations. For that purpose, one needs the corresponding Jacobian. This matrix is given by a part of the system Jacobian and can be evaluated using again either standard higher-order automatic differentiation or the more efficient variants discussed above. However, the initialization of the input Taylor coefficients is considerably easy since one simply has to choose the corresponding unit vectors.

Once a consistent point at time $t = 0$ is computed, one may for example apply an explicit Taylor method to integrate the DAE. For that purpose, only one solve of a linear system with the system Jacobian J as linear operator is required for each order of the Taylor method. Hence, one performs one LU-factorization of J. Subsequently, one only has to evaluate higher-order derivatives occurring in the right-hand sides. For this purpose, the technique explained in the preceding section can be used.

4 Implementation Details

For simplicity we have assumed that the DAE system has been written in autonomous form, but an explicit time dependence could certainly be accounted for too. Although variations are possible we suggest that the problem be specified by an evaluation code of the following form

```
void sys_eval(int n, adouble** y, adouble* F)
```

using the active variable type adouble provided by ADOL-C. Here y[j][k] represents $y_{j,k}$, i.e. the k-th derivative of the j-th variable with $k \le m_j$. In other words, the calling program must have allocated n adouble pointers y[j] for $j = 0, \ldots, n-1$ where each of them is itself a vector of $m_j + 1 =:$ m[j] adoubles. On exit the components F[i] for $i = 0, \ldots, n-1$ contain the function components F_i in Pryce notation.

Provided the code sys_eval does not contain any branches it must be called only once before the actual DAE solving begins. Before the call, y and F must be allocated and y initialized by a loop of the form

```
int tag = 1;
trace_on(tag)
  y = new adouble*[n]; F = new adouble[n];
  for(j=0; j<n; j++)
  {  y[j] = new adouble[m[j]];
     for (i=0; i<m[j]; i++)
       y[j][i]<<=yp[j][i];
  }
  sys_eval(n, y, F)
  for(j=0; j<n; j++)
    F[j] >>= Fp[j]
trace_off()
```

Here, yp[j][i] is an array of double values at which sys_eval can be sensibly called. The loop after the call to sys_eval determines the dependent variables in ADOL-C terminology, where Fp[j] is like yp[j][i] of type double. Now the DAE system has been *taped*. If the function evaluation contain branches, the generated tape can be reused as long as the control does not change. If the control flow changes, the return values of the drivers for computing the desired derivatives indicate that the tape is no longer valid and a retaping has to be performed. Hence, by monitoring the corresponding return values the correctness of the derivative information can be ensured while keeping the effort for the taping as low as possible.

Once, the tape is generated, function and derivative evaluations are performed for example by the routines zos_forward_partx(..), fos_forward_partx(..), hos_forward_partx(..), and jacobian_partx(..) that are problem independent. For example the call

```
zos_forward_partx(tag,n,n,m,yp,Fp)
```

will yield as output the system values Fp[j] for arbitrary inputs yp[j][i] and the array m describing the partition of yp. Here, zos_forward stands for **z**ero-**o**rder **s**calar forward mode, since no derivatives are required.

Now suppose we have allocated and assigned values to a three dimensional tensor yt[j][i][r] for $j < n, i < m[j], r \leq b$. Mathematically, this is interpreted as the family of Taylor expansions

$$\text{y[j][i]} \equiv \sum_{r=0}^{b} \text{yt[j][i][r]}\; t^r \quad .$$

For ADOL-C, the values y[j][i][r] are completely independent but for use in the integration method proposed by Pryce they must be given values that are consistent in that

$$\text{yt [j][i][r]} = y_{j,i+r}/r! \quad .$$

Here the $y_{j,i+r} = y_i^{(i+r)}$ are the already known or guessed solution values and derivatives. All derivatives of higher-order should be set to zero. Now the call

```
fos_forward_partx(tag,n,n,m,yt,Ft)
```

yields the Taylor coefficients Ft[j][r] for $r < 2$, where fos_forward stands for **f**irst-**o**rder **s**calar forward mode and the call

```
hos_forward_partx(tag,n,n,m,b,yt,Ft)
```

yields the Taylor coefficients Ft[j][r] for $r \leq b$ of the resulting expansion

$$\text{F[i]} \equiv \sum_{r=0}^{b} \text{Ft[i][r]}\, t^r.$$

In other words, the derivative $(r!)$ Ft[j][r] corresponds exactly to the value $F_{j,r}$ needed for the approach described in [10,11] by Pryce, except for the scaling by $r!$ that has to be done by the user.

For computing the system Jacobian that is also required by the method of Pryce and similar integration methods, ADOL-C provides also a special driver. For using it one must allocate the array jac[i][j][k] for $i < n, j < n, k \leq m_j$. In order to obtain the values

$$\text{jac[i][j][k]} \equiv \partial F_i / \partial y_{j,k}$$

of this Jacobian, the user has to call the new Jacobian driver

```
jacobian_partx(tag,n,m,n,xp,jac).
```

of ADOL-C. The presented new driver of ADOL-C are available in the current version 1.9.0 and have been incorporated into a software-prototype in order to test the calculation of higher order derivatives for the integration of high-index DAEs. The achieved numerical results are presented in the next section.

5 Numerical Example

We implemented a very simple version of the algorithm given in [10] to illustrate the capabilities of ADOL-C to provide the required higher-order derivatives. Furthermore, the resulting code may form one possibility to verify the run-time saving that can be achieved using the improvements stated in Section 2.

As test example, we choose a model of two pendula from [10], where the λ component of the first one controls the length of the second one. This system with index 5 is described by the DAE

$$\begin{aligned} F_1 &= x'' + x\lambda \;=\; 0 & \qquad F_4 &= u'' + u\kappa \;=\; 0 \\ F_2 &= y'' + y\lambda - g \;=\; 0 & \qquad F_5 &= v'' + v\lambda - g \;=\; 0 \\ F_3 &= x^2 + y^2 - L \;=\; 0 & \qquad F_6 &= u^2 + v^2 - (L + c\lambda)^2 \;=\; 0 \end{aligned}$$

and has four degrees of freedom. For the numerical tests presented below, we use the gravity constant $g = 1$, the length $L = 1$ of the first pendulum, $c = 0.1$ and simulate the behaviour of both pendula for the time interval $[0, 40]$. This choice of T ensures that we simulate the pendula for a period where they do not show chaotic behaviour, which

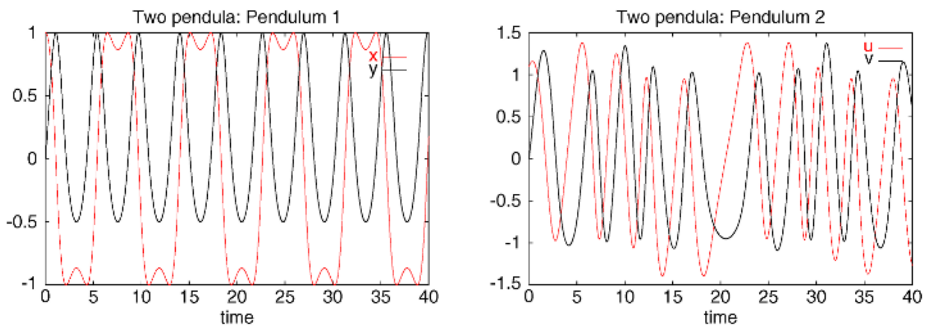


Fig. 1. Index 5 two-pendulum problem, numerical results

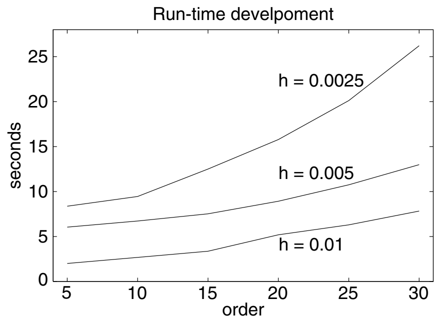


Fig. 2. Index 5 two-pendulum problem, run-times

is the case for $T \geq 50$, cf. [10]. The values $(x_0, y_0) = (u_0, v_0) = (1, 0)$ and $(x_0', y_0') = (u_0', v_0') = (0, 1)$ serve as initial point. In order to judge the influence of the derivative calculation on the run-time, we consider three discretizations based on the step size $h = 0.0025, 0.005, 0.01$ which results in the time step numbers $16\,000, 8\,000$ and $4\,000$ respectively. In addition we use explicit Taylor methods of order $5, 10, 15, 20, 25, 30$ for the integration of the DAE.

The computed results for the two pendula are illustrated by Fig. 1, where the first one shows obviously a periodic behaviour. Since the length of the second one varies in dependence on the first pendulum one can observe for the second one a non-periodic solution which results eventually in a chaotic behaviour. However, for the chosen combinations of step size and integration order the computed solutions are identical for the first pendulum and close or at least comparable for the second one. This fact was acceptable for us since the influence of the derivative order on the computing time was the main subject.

The simulations were computed using a Red Hat Linux system, with an AMD Athlon XP 1666 Mhz processor and 512 MB RAM. The required computing times for the different step sizes and integration orders are illustrated by Fig. 2. Here, for one specific step size the computational efforts varies mainly due to the integration order of the explicit Taylor method. The predicted nonlinear behaviour can be observed very clearly for the step size $h = 0.0025$. For the larger time steps, i.e. $h = 0.01$ and $h = 0.005$, the derivative calculation is dominated by the linear algebra cost. Therefore, Fig. 2 shows only a slight nonlinear influence of the integration order on the run-time. Nevertheless, the numerical experiment confirms that the effort for computing higher-order derivatives increases only moderately when using the AD-tool ADOL-C. This is in accordance to the theoretical results sketched in Section 2.

6 Conclusion and Outlook

This paper discusses the computation of higher-order derivatives using automatic differentiation in the context of high index DAEs. For that purpose, the standard higher-order forward sweep of automatic differentiation based on Taylor arithmetic is discussed as well as two possible improvements that are valuable for very high order derivatives. This methodology is embedded in the structural analysis for high-index DAEs. One case study presents numerical results achieved with the AD-tool ADOL-C that confirm the theoretical complexity of computing higher-order derivatives.

Certainly, the implementation of the improvements for the generation of Taylor expansions presented in this paper forms one specific future challenge. This may include also a possibly thread-based parallelization. Here one can exploit the coarse-grained nature of Taylor computations that differ significantly from the situation when computing first-order derivatives using AD. An additional task is to ease the use of existing AD-tools for the usage in connection with the structural approach to integrate high-index DAEs. For that purpose, we presented a description of new drivers provided by ADOL-C 1.9.0 that take into account the special structure of a given DAE system where in addition to the values of the variables also the values of specific derivatives of the variables enter the evaluation of the system function.

References

1. C. Bendtsen and O. Stauning: FADBAD, a flexible C++ package for automatic differentiation. Department of Mathematical Modelling, Technical University of Denmark, 1996.
2. R. Brent and H. Kung: Fast algorithms for manipulating formal power series. Journal of the Association for Computing Machinery **25**, 581–595, 1978.
3. F. Cellier and H. Elmquist: Automated formula manipulation supports object-oriented continuous-system modelling. IEEE Control System Magazine **13**, 28–38, 1993.
4. http://www.seanet.com/~bradbell/CppAD/
5. Y.F. Chang and G. Corliss: Solving ordinary differential equations using Taylor series. ACM Trans. Math. Software **8**, 114–144, 1982.
6. W. Feehery and P. Barton: Dynamic optimization with state variable path constraints. Comput. Chem. Engrg. **22**, 1241–1256, 1998.
7. A. Griewank: *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*, Frontiers in Appl. Math. 19, SIAM, Phil., 2000.
8. A. Griewank, D. Juedes, and J. Utke: ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. TOMS **22**, 131–167, 1996.
9. C.C. Pantelides: The consistent initialization of differential-algebraic systems. SIAM J. Sci. Statist. Comput. **9**, 213–231, 1988.

10. J. Pryce: Solving high-index DAEs by Taylor series. Numer. Algorithms **19**, 195–211, 1998.
11. J. Pryce: A simple structural analysis method for DAEs. BIT **41**, 364–394, 2001.
12. G. Reißig, W. Martinson, and P. Barton: Differential-algebraic equations of index 1 may have an arbitrarily high structural index. SIAM J. Sci. Comput. **21**, 1987–1990, 2000.

Edge-Disjoint Hamiltonian Cycles of WK-Recursive Networks

Chien-Hung Huang[1], Jywe-Fei Fang[2], and Chin-Yang Yang[3]

[1] National Formosa University , Yun-Lin 632, Taiwan, R.O.C.
[2] St.John's and St.Mary's Institute of Technology, Taipei 251, Taiwan, R.O.C.
[3] National Taiwan University Hospital, Taipei 100, Taiwan, R.O.C.

Abstract. In this paper, we show that there exist n edge-disjoint Hamiltonian cycles in the WK-Recursive networks with amplitude $2n+1$. By the aid of these edge-disjoint Hamiltonian cycles, nearly optimal all-to-all broadcasting communication on all-port WK-Recursive networks can be derived.

1 Introduction

In massively parallel MIMD systems, the topology plays a crucial role in issues such as communication performance, hardware cost, potentialities for efficient applications and fault tolerance capabilities. A topology named $WK\text{-}Recursivenetwork$ has been proposed by Vecchia and Sanges under CAPRI (Concurrent Architecture and Programming environment for highly Integrated systems) project supported by the Strategic Program on Parallel Computing of the National Research Council of Italy [1]. The topology has many attractive properties, such as high degree of regularity, symmetry and efficient communication. Particularly, for any specified number of degree, it can be expanded to arbitrary size level without reconfiguring the links. WK-Recursive networks have received considerable attention. Researchers have devoted themselves to various issues of WK-Recursive networks such as broadcasting algorithms [2], topological properties [6], substructure allocation [5] and communication analysis [4]. d'Acierno et al. built a neural network on the WK-Recursive network, which gained considerable performance enhancement [15]. In recent years, Fang et al. have proposed a novel broadcasting scheme for the WK-Recursive network, which requires only constant data included in each message and constant time to determine the neighbors to forward the message [12]. Fu has discussed the $Hamiltonian\text{-}connected$ property of the WK-Recursive network [13]. Wu et al. have proposed a generalized and efficient allocation scheme to the Recursively Decomposable Interconnection Networks, which is including the WK-Recursive network [14].

A graph G is said to be $Hamiltonian\text{-}decomposable$ if its edge set can be partitioned into edge-disjoint Hamiltonian cycles. Thus, some communication problems, such as many-to-many $broadcasting$ and many-to-many $scattering$ in $all\ port\ model$, can be nearly optimally solved [8]. In fact, many researchers have studied this issue of various topologies [9, 10]. To the best of our knowledge, there exists no article addressing this issue of WK-Recursive networks. In this paper, we show that WK-Recursive networks with amplitude $2n+1$ are Hamiltonian-decomposable.

2 Notations and Background

A *complete graph* with n nodes, denoted by K_n, is a graph in which every two distinct nodes are adjacent. A WK-Recursive network with amplitude W and level L, denoted by WK(W, L), can be recursively constructed as: WK(W, 1) is a K_W in which each node has one free link and $W - 1$ links that are used for connecting to other nodes. Clearly, WK(W, 1) has W nodes and W free links. WK(W, C) consists of W copies of WK(W, $C - 1$) as supernodes and the W supernodes are connected as a K_W, where $2 \leq C \leq L$. By induction, it is easy to see that WK(W, L) has W^L nodes and W free links. Consequently, for any specified number of degree W, WK-Recursive networks can be expanded to arbitrary level L without reconfiguring the links. In Fig. 1, the structures of WK(5, 1) and WK(5, 2) are shown.

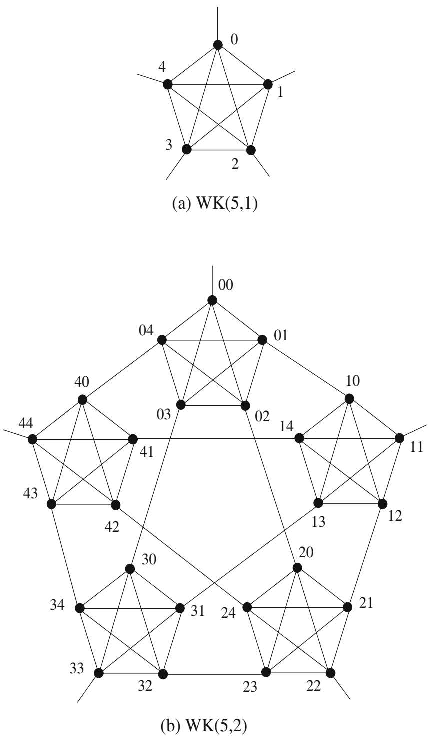


Fig. 1. The structures of WK(5, 1) and WK(5, 2)

The following addressing scheme for WK(W, L) is described in [7]. After fixing an origin and an orientation (clockwise or counterclockwise), within each WK(W, 1) subnetwork, a node is labeled with an index digit d_1 from 0 to $W - 1$. Similarly, within each WK(W, C) subnetwork, a WK(W, $C - 1$) subnetwork is labeled with an index d_C from 0 to $W - 1$, where $2 \leq C \leq L$. Hence, each node of WK(W, L) is labeled with an unique address $d_L d_{L-1} \ldots d_2 d_1$ as illustrated in Figure. 1. Likewise, A subnetwork of

WK(W, L) can be represented by a string of L symbols over set $\{0,1,...,W-1\}\smile\{*\}$, where * is a "don't care" symbol. That is, each WK(W, C) subnetwork of WK(W, L) can be denoted by $d_L d_{L-1} \dots d_{C+1}(*)^C$, where $(*)^(C)$ represents C consecutive *'s. For example, in WK(5, 3), 0** is the subnetwork $\{0d_2d_1|0 \le d_2 \le 4$ and $0 \le d_1 \le 4\}$.

For a subnetwork

$$d_L d_{L-1} \dots d_{C+1}(*)^C \text{ in WK}(W, L),$$

a node with address $d_L d_{L-1} \dots d_{C+1}(d_C)^C$ is called a corner node of $d_L d_{L-1} \dots d_{C+1}(*)^C$. For example, in WK(5, 3), 000, 011, 022, 033 and 044 are corner nodes of 0**. Specifically, the node $d_L d_{L-1} \dots d_{C+1}(d_C)^C$ is called the $d_C - corner$ of $d_L d_{L-1} \dots d_{C+1}(*)^C$. For example, in WK(5, 3), 022 is called 2-corner of 0**. In this paper, a link within a WK(W, 1) subnetwork is called an *inner-cluster link.*

Definition 1. The inner-cluster links of node $d_L d_{L-1} \dots d_2 d_1$ are defined as $(d_L d_{L-1} \dots d_2 d_1, d_L d_{L-1} \dots d_2 h)$, where $0 \le h \le W-1$ and $d_1 \ne h$.

For example, in WK(5, 3), (002, 000), (002, 001), (002, 003) and (002, 004) are inner-cluster links of node 002. Clearly, each node has $W-1$ inner-cluster links in WK(W, L). A link connecting two WK(W, C) subnetworks, where $1 \le C \le L-1$, is called an *inter-cluster link* and specifically a *C-level link.*

Definition 2. the C level inter-cluster link of node $d_L d_{L-1} \dots d_{C+1}(d_C)^C$, where $d_{C+1} \ne d_C$, is defined as $(d_L d_{L-1} \dots d_{C+1}(d_C)^C, d_L d_{L-1} \dots d_C(d_{C+1})^C)$.

For example, in WK(5, 3), (022, 200) is a 2-level link and (012, 021) is a 1-level link. Note that each node except the corner nodes $(d_L)^L$, where $0 \le d_L \le W-1$, has exactly one inter-cluster link in WK(W, L). Each corner node $(d_L)^L$ of WK(W, L) has no inter-cluster link but a free link. In this paper, the outline graph of WK(W, L) is to take each WK(W, 1) subnetwork as a supernode. Since WK(W, L) can be constructed recursively, we have the following proposition. Proposition 1. The outline graph of WK(W, L) is WK(W, $L-1$).

3 The Results

In this section, we show how to partition the edge set of WK($2n+1$, L) into n edge-disjoint Hamiltonian cycles. First, we show that the result is true for $L = 1$. According to the definition, WK($2n+1$, 1) is K_{2n+1}. It is well-known that K_{2n+1} can be decomposed into n Hamiltonian cycles[9]. Let these $2n+1$ nodes of K_{2n+1} be labeled as v_0, v_1, K, v_{2n}. H_i, where $0 \le i \le n-1$, is defined as follows:

$v_{2n}, v_i, v_{i+1}, v_{i-1}, v_{i+2}, v_{i-2}, \dots, v_{n+i-1}, v_{n+i+1}, v_{n+i}, v_{2n}$

All subscripts except $2n$ are positive and expressed modulo $2n$. H_i, where $0 \le i \le n-1$, is a Hamiltonian cycle of K_{2n+1} and these n Hamiltonian cycles are edge-disjoint [9]. Thus we can state the following lemma.

Lemma 1. The edge set of K_{2n+1} can be partitioned into n edge-disjoint Hamiltonian cycles. HP_i, where $0 \le i \le n-1$, is defined as follows:

$v_i, v_{2n}, v_{n+i}, v_{n+i+1}, v_{n+i-1}, \dots, v_{i-2}, v_{i+2}, v_{i-1}, v_{i+1}$ (for i is even)

$v_{n+i}, v_{2n}, v_i, v_{i+1}, v_{i-1}, v_{i+2}, v_{i-2}, \dots, v_{n+i-1}, v_{n+i+1}$ (for i is odd)

All subscripts except $2n$ are positive and expressed modulo $2n$. Clearly, HP_i, where $0 \le i \le n-1$, is a Hamiltonian path of K_{2n+1} and these n Hamiltonian paths are edge-disjoint. Moreover, each source node and destination node of HP_i, where $0 \leq i \leq n-1$, are disjoint each other. The source node and destination node of HP_i are (v_0, v_1), (v_2, v_3), (v_4, v_5),. . .,(v_{2m}, v_{2m+1}) for i= 2, 4, 6,. . ., $2m$, where $2m \leq n-1$. The source node and destination node of HP_i are (v_{n+1}, v_{n+2}), (v_{n+3}, v_{n+4}), (v_{n+5}, v_{n+6}),. . .,(v_{n+2m+1}, v_{n+2m+2}) for i = 1, 3, 5,. . ., $2m+1$, where $2m+1 \leq n-1$ and all subscripts except $2n$ are positive and expressed modulo $2n$. Now we show that there exist n edge-disjoint Hamiltonian cycles in the WK-Recursive networks with amplitude $2n+1$.

Theorem 2. WK($2n+1$, L) contains n edge-disjoint Hamiltonian cycles.

Proof. We will prove the theorem by induction on L. For L =1, WK($2n+1$, 1) is K_{2n+1}. According to Lemma 1, the result is true for L = 1. Hypothesis: Assume that WK($2n+1$, k) contains n edge-disjoint Hamiltonian cycles. Induction Step: Because the outline graph of WK($2n+1$, $k+1$) is WK($2n+1$, k). By hypothesis, WK($2n+1$, k) contains n edge-disjoint Hamiltonian cycles. Let $HC(k, i)$, where $1 \leq i \leq n$, be n edge-disjoint Hamiltonian cycles in the outline graph of WK($2n+1$, $k+1$). Clearly, each WK($2n+1$,1) subnetwork V_j*, where $0 \leq j \leq (2n+1)^k - 1$, is incident to exactly two edges in $HC(k, i)$. That is, for each WK($2n+1$,1) subnetwork V_j*, there are exactly two nodes whose inter-cluster links are contained in $HC(k, i)$. We label these two nodes as $v_{j,i}$ and $v_{j,i+1}$ ($v_{j,n+i}$ and $v_{j,n+i+1}$) for each i is even(odd), where $1 \leq i \leq n$ and the second subscripts except $2n$ are all positive and expressed modulo $2n$. In each WK($2n+1$,1) subnetwork V_j*, the node which is not incident to an inter-cluster link in all $HC(k, i)$ is labeled as $v_{j,2n}$. For each WK($2n+1$,1) subnetwork V_j*, where $0 \leq j \leq (2n+1)^k - 1$ and $1 \leq i \leq n$, we define $HP_{j,i}$ as follows:

$v_{j,i}, v_{j,2n}, v_{j,n+i}, v_{j,n+i+1}, v_{j,n+i-1}, \ldots, v_{j,i-2}, v_{j,i+2}, v_{j,i-1}, v_{j,i+1}$ (for i is even)
$v_{j,n+i}, v_{j,2n}, v_{j,i}, v_{j,i+1}, v_{j,i-1}, v_{j,i+2}, v_{j,i-2}, \ldots, v_{j,n+i-1}, v_{j,n+i+1}$ (for i is odd)

The second subscripts except $2n$ are all positive and expressed modulo $2n$. Clearly, $HP_{j,i}$ is a Hamiltonian path of WK($2n+1$,1) subnetwork V_j* and these n Hamiltonian paths are edge-disjoint. For each $0 \leq i \leq n-1$, let $HC(k+1, i) = HC(k, i) \smile HP_{0,i} \smile HP_{1,i} \smile K \smile HP_{N-2,i} \smile HP_{N-1,i}$, where $N = (2n+1)^k$. $HC(k+1, i)$ is a Hamiltonian cycle of WK($2n+1$, $k+1$) and these n Hamiltonian cycles are edge-disjoint. This extends the induction and completes the proof. □

For example, edges of WK(5, 2) can be partitioned into Hamiltonian cycles as follows. First, according to Lemma 1, WK(5, 1) can be partitioned into Hamiltonian cycles as shown in Figure. 2(a). Note that the outline graph of WK(5, 2) is WK(5, 1). If each WK(5, 1) subnetwork is regarded as a supernode, the inter-cluster links of WK(5, 2) can be partitioned into two edge-disjoint Hamiltonian cycles. Clearly, K_5 is node symmetric. Thus, we can label the nodes of each WK(5, 1) subnetwork according to proof of Theorem 2. Consequently, edges of WK(5, 2) can be partitioned into two edge-disjoint Hamiltonian cycles as illustrated in Figure. 2(b).

4 Conclusions

In this paper, we have shown that WK-Recursive networks with amplitude $2n+1$ contains n edge-disjoint Hamiltonian cycles. By the aid of these edge-disjoint Hamiltonian cycles, nearly optimal all-to-all broadcasting communication on all-port WK-Recursive

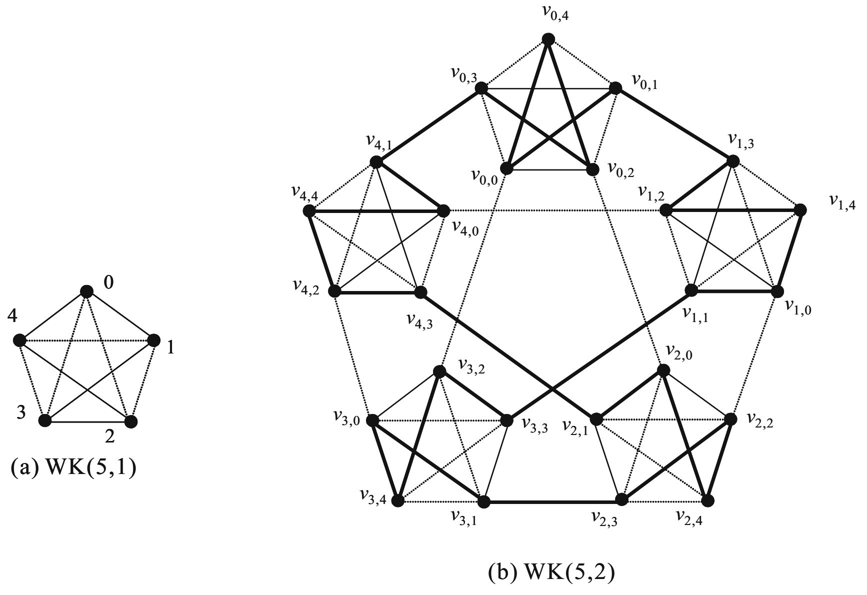


Fig. 2. Hamiltonian decomposition of WK(5, 2)

networks can be derived. However, the issue of one-port WK-Recursive networks is still open.

This work was supported in part by the National Science Council of the Republic of China under the contract number: NSC92-2213-E-129-010.

References

1. G. D. Vecchia and C. Sanges, “A recursively scalable network VLSI implementation,“ *Future Generat. Comput. Syst.* vol. 4, pp. 235-243, 1988.
2. G. D. Vecchia and C. Sanges, “An optimal broadcasting technique for WK-Recursive topologies,“ *Future Generat. Comput. Syst.* vol. 5, pp. 353-357, 1989/1990.
3. F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays,Trees, Hypercubes,* California: Mogran Kaufmann, 1992.
4. R. Fernandes, “Recursive interconnection networks for multicomputer networks,“ *Proceed. Int. Conf. Parallel Process.,* vol. 1, pp. 76-79, 1992.
5. R. Fernandes and A. Kanevsky, “Substructure allocation in recursive interconnection networks,“ *Proceed. Int. Conf. Parallel Process.,* vol. 1, pp. 315-318, 1993.
6. A. I. Mahdaly, H. T. Mouftah, N. N. Hanna, “Topological properties of WK-recursive networks," *Proceed. Second IEEE Workshop on Future Trends of Distributed Computing Systems,* pp. 374-380, 1990.
7. G. D. Vecchia and C. Sanges, “Recursively scalable network for message passing architecture," *Proceed. Int. Conf. Parallel Processing and Applications,* vol. 1, pp. 33-40, May 1987.
8. S. L. Johnsson and C. T. Ho, “Optimum broadcasting and personalized communication in hypercubes," *IEEE Transaction on Computers,* vol. 38, pp. 1249-1268,1989.

9. G. Chartrand and O. R. Oellermann, *Applied and Algorithmic Graph Theory,* McGraw-Hill,1993.
10. D. Barth and A. Raspaud, "Two edge-disjoint Hamiltonian cycles in the butterfly graph," *Inform. Process. Lett.,* vol. 51, pp. 175-179,1994.
11. J. C. Bermond, O. Favaron and M. Maheo, "Hamiltonian decomposition of Cayley graphs of degree four," *J. Combin. Theory,* vol. 46, pp. 142-153, 1989.
12. J. F. Fang, G. J. Lai, Y. C. Liu and S. T. Fang, "A novel broadcasting scheme for WK-Recursive networks," *Proceed. IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing,* pp. 1028-1031, 2003.
13. J. S. Fu, "Hamiltonian-connectedness of the WK-recursive network," *Proceed. Int. Conf. Symposium on Parallel Architectures, Algorithms and Networks,* pp.569-574, 2004.
14. F. Wu and C. C. Hsu, "A generalized processor allocation scheme for recursively decomposable interconnection networks," *IEICE Transactions on Information and Systems,* vol. E85-D, pp. 694-713, 2002.
15. A. d'Acierno, R. Del Balio, G. De Pietro and R. Vaccaro, "A parallel simulation of fully connected neural networks on a WK-recursive topology," *Proceed. IEEE International Joint Conference on Neural Networks,* pp. 1850-1854, 1991.

Simulation-Based Analysis of Parallel Runge-Kutta Solvers

Matthias Korch and Thomas Rauber

University of Bayreuth, Department of Mathematics, Physics, and Computer Science
{matthias.korch,rauber}@uni-bayreuth.de

Abstract. We use simulation-based analysis to compare and investigate different shared-memory implementations of parallel and sequential embedded Runge-Kutta solvers for systems of ordinary differential equations. The results of the analysis help to provide a better understanding of the locality and scalability behavior of the implementations and can be used as a starting point for further optimizations.

1 Introduction

The modeling of many scientific and engineering problems leads to systems of ordinary differential equations (ODEs). The numerical solution of such problems requires large amounts of computational resources, particularly if the ODE system is large or the evaluation of the right-hand-side function is expensive. We aim at the development of new, fast realizations of parallel ODE solvers that efficiently exploit the memory hierarchy of current microprocessors. We start with an analysis of existing parallel and sequential methods that assists in the detection of scalability bounds and further sources of performance degradation. The paper considers the simulation-based analysis of embedded Runge-Kutta (RK) methods. In particular, we study locality optimizations for general embedded RK methods based on program transformations similar to [11], and locality optimization and parallelization using pipelining to exploit the characteristic access structure of an important class of ODEs resulting from the spatial discretization of partial differential equations (PDEs) [6]. In contrast to [6], the parallel realizations are based on a multi-threaded realization using Pthreads.

Investigations on real computer systems—as presented in [6,11]—show that there are several limitations when performing a software analysis on real systems. For example, only a limited number of hardware events is measurable as provided by the manufacturer of the system, the execution of the program to be analyzed may be disturbed by other user processes competing for shared resources, and the measurement itself may influence its outcome if additional code needs to be inserted into the program or if the program has to be interrupted in order to read the state of the machine. Additional experiments using simulators can help overcome these problems. Particularly promising are instruction-set-level simulators which enable full-system simulation of real parallel architectures and provide a wide range of instrumentation facilities. We use Simics [9] for analyzing the locality behavior and scalability properties of several sequential and parallel multi-threaded versions of embedded RK solvers. Simics has already been used successfully to investigate the cache memory behavior of PDE solvers [13]. We present experiments performed on a simulated SPARC V9 ISA, and compare our results with experiments performed on a real Sun Fire server. As examples we use two sparse ODE systems resulting from applying the method of lines to time-dependent PDEs.

The rest of the paper is organized as follows: Section 2 describes the ODE solution methods considered. Section 3 shows the ODE test set used, and Section 4 gives an overview of the different implementations. Section 5 presents simulation experiments and provides a comparison with runtime tests on a real system. Section 6 discusses related work, and Section 7 concludes the paper.

2 Solution of Initial Value Problems Using Embedded Runge-Kutta Methods

In this paper, we consider the solution of initial value problems (IVPs)

$$\boldsymbol{y}'(t) = \boldsymbol{f}(t, \boldsymbol{y}(t)) \ , \quad \boldsymbol{y}(t_0) = \boldsymbol{y}_0 \ , \quad \boldsymbol{y} : \mathbb{R} \to \mathbb{R}^n \ , \quad \boldsymbol{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n \ , \tag{2.1}$$

of systems of ODEs. Such systems can be solved efficiently by explicit RK methods with stepsize control using embedded solutions. An embedded RK method with s stages uses the stage vectors $\boldsymbol{v}_1, \dots, \boldsymbol{v}_s$ to compute two new approximations $\eta_{\kappa+1}$ and $\hat{\eta}_{\kappa+1}$ from the two previous approximations η_κ and $\hat{\eta}_\kappa$. This is represented by the computation scheme

$$\begin{aligned} \boldsymbol{v}_l &= \boldsymbol{f}(x_\kappa + c_l h_\kappa \ , \eta_\kappa + h_\kappa \sum_{i=1}^{l-1} a_{li} \boldsymbol{v}_i) \ , \qquad l = 1, \dots, s \ , \\ \eta_{\kappa+1} &= \eta_\kappa + h_\kappa \sum_{l=1}^{s} b_l \boldsymbol{v}_l \ , \qquad \hat{\eta}_{\kappa+1} = \eta_\kappa + h_\kappa \sum_{l=1}^{s} \hat{b}_l \boldsymbol{v}_l \ . \end{aligned} \tag{2.2}$$

The coefficients a_{ij}, c_i, b_i, and $\hat{b}_i$ are determined by the RK method used. A straightforward implementation of (2.2) leads to the program shown in Fig. 1. It contains a non-tightly nested loop structure consisting of a loop over the stage vector computation with the computation of the argument vector $\boldsymbol{w}$ as inner loop and the loop over vector components as innermost loop. Because the function evaluations of the right-hand-side function $\boldsymbol{f}$ may access all components of $\boldsymbol{w}$, two barriers must be used in the data-parallel implementation to ensure that the computation of $\boldsymbol{w}$ has been finished before $\boldsymbol{f}$ is evaluated and that $\boldsymbol{w}$ is not modified before the function evaluation is complete.

3 ODE Test Set

Our first test problem is the two-dimensional Brusselator equation (BRUSS2D)

$$\begin{aligned} \frac{\partial u}{\partial t} &= 1 + u^2 v - 4.4u + \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right), \\ \frac{\partial v}{\partial t} &= 3.4u - u^2 v + \alpha \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) \end{aligned} \tag{3.3}$$

```
// compute the stage vectors $v_1, \ldots, v_s$

for ($j = first$; $j \le last$; $j$++) $\mathbf{v}_1[j] = f_j(x + c_1 h, \eta)$;
for ($i = 2$; $i \le s$; $i$++)
{
    barrier();
    for ($j = first$; $j \le last$; $j$++) $\boldsymbol{w}[j] = a_{i1}\mathbf{v}_1[j]$;
    for ($l = 2$; $l < i$; $l$++)
        for ($j = first$; $j \le last$; $j$++) $\boldsymbol{w}[j]$ += $a_{il}\mathbf{v}_l[j]$;
    for ($j = first$; $j \le last$; $j$++) $\boldsymbol{w}[j] = h\boldsymbol{w}[j] + \eta[j]$;
    barrier();
    for ($j = first$; $j \le last$; $j$++) $\mathbf{v}_i[j] = f_j(x + c_i h, \boldsymbol{w})$;
}

// compute $\mathbf{u} = \eta_{\kappa+1} - \eta_\kappa$ and $\mathbf{t} = \hat{\eta}_{\kappa+1} - \eta_{\kappa+1}$ (using $\tilde{\boldsymbol{b}} = \hat{\boldsymbol{b}} - \boldsymbol{b}$)

for ($j = first$; $j \le last$; $j$++) { $\mathbf{u}[j] = b_1\mathbf{v}_1[j]$; $\mathbf{t}[j] = \tilde{b}_1\mathbf{v}_1[j]$; }
for ($i = 2$; $i \le s$; $i$++)
    for ($j = first$; $j \le last$; $j$++) { $\mathbf{t}[j]$ += $\tilde{b}_i\mathbf{v}_i[j]$; $\mathbf{u}[j]$ += $b_i\mathbf{v}_i[j]$; }

use $\mathbf{t}$ to perform error control and stepsize selection
use $\mathbf{u}$ to update $\eta_{\kappa+1}$ if step is accepted
```

Fig. 1. Program fragment of one time step of parallel implementation (A)

where the unknown functions u and v represent the concentrations of two substances in the domain $0 \le x \le 1$, $0 \le y \le 1$ [4]. A Neumann boundary condition $\partial u/\partial \eta = 0, \partial v/\partial \eta = 0$, and the initial conditions $u(x, y, 0) = 0.5 + y$, $v(x, y, 0) = 1 + 5x$ are used. A standard five-point-star discretization of the spatial derivatives on a uniform $N \times N$ grid leads to a sparse ODE system of dimension $n = 2N^2$ for the discretized solution $\{U_{ij}\}_{i,j=1,\ldots,N}$ and $\{V_{ij}\}_{i,j=1,\ldots,N}$

$$\begin{aligned} \frac{dU_{ij}}{dt} &= 1 + U_{ij}^2 V_{ij} - 4.4U_{ij} + \alpha(N-1)^2 \\ &\quad \times (U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j}), \\ \frac{dV_{ij}}{dt} &= 3.4U_{ij} - U_{ij}^2 V_{ij} + \alpha(N-1)^2 \\ &\quad \times (V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1} - 4V_{i,j}), \end{aligned} \tag{3.4}$$

where a constant number of components of the argument vector are accessed in each evaluation of the right-hand-side function. This is a non-stiff system for $\alpha = 2 \cdot 10^{-4}$ and small values of N.

The second problem considered is the Medical Akzo Nobel problem (MEDAKZO) [8]. This problem originates from the penetration of radio-labeled antibodies into a tissue that has been infected by a tumor. It consist of two PDEs

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} - kuv \ , \qquad \frac{\partial v}{\partial t} = -kuv \ . \tag{3.5}$$

Semi-discretization leads to an ODE system of the form

$$\begin{aligned} \boldsymbol{y}'(t) &= \boldsymbol{f}(t,\boldsymbol{y}) \ , \quad \boldsymbol{y}(0) = \boldsymbol{g} \ , \qquad \boldsymbol{y} \in \mathbb{R}^{2N} \ , \qquad \text{where} \\ f_{2j-1} &= \alpha_j \frac{y_{2j+1} - y_{2j-3}}{\Delta\zeta} + \beta_j \frac{y_{2j-3} - 2y_{2j-1} + y_{2j+1}}{(\Delta\zeta)^2} - k\, y_{2j-1} y_{2j} \ , \\ f_{2j} &= -k\, y_{2j} y_{2j-1} \ . \end{aligned} \tag{3.6}$$

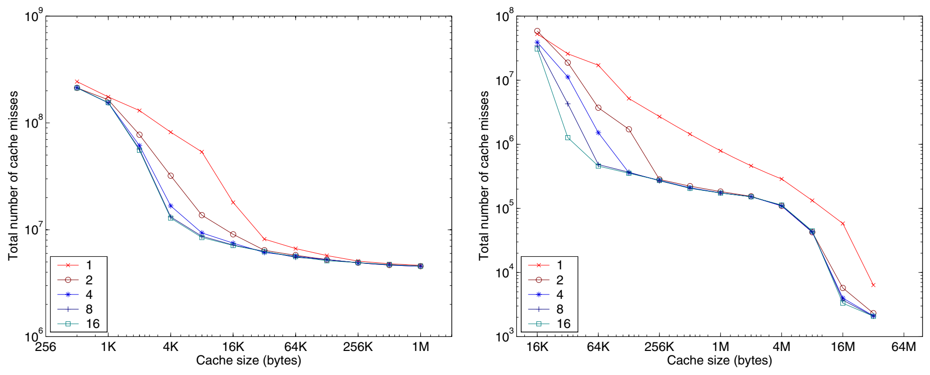
As the Brusselator equation, this ODE system is sparse, i.e., the function evaluations only access a small, constant number of components of the argument vector. A detailed derivation of (3.6) together with initial and boundary conditions can be found in [8].

For the investigation of the locality behavior, the system size and the number of stages of the RK method should be as large as possible such that not all memory locations accessed during one time step can be stored in the cache and a good locality behavior is profitable. Therefore, we use the grid size $N = 250$ (resulting in 125000 components) together with DOPRI5(4) (7 stages) for BRUSS2D, and $N = 15000$ (resulting in 30000 components) together with DOPRI8(7) (13 stages) in the experiments with MEDAKZO. Refining the mesh size to increase the system size, however, also increases the degree of stiffness of the problems. Therefore, RK methods of lower order would be more suitable for a practical numerical solution. However, a high number of stages is required for the creation of large working sets (cf. Section 5) since the only alternative is to increase the system size and thus to reduce the mesh size even further.

4 Overview of the Implementations

To develop efficient parallel implementations of embedded RK solvers, we derive several different implementation variants by applying modifications to the loop structure and taking the special dependences of particular ODE systems into account. The modifications of the loop structure increase the locality and allow a better utilization of the memory hierarchy. On shared-memory machines a higher locality usually increases the scalability since the memory operations issued by the processors can be satisfied from one of the local caches more often, and the traffic on the system interconnect is reduced. Additionally, the loop transformations may enable more efficient synchronization. The implementations investigated include general implementations that can be used with arbitrary ODEs, and pipelining implementations which are specialized in sparse ODE systems with a limited access distance of the function evaluations.

General Implementations. The general implementations have been derived starting from the straightforward implementation (A) (Fig. 1) by applying loop transformation techniques such as loop interchange and loop fusion. Further, the data structures have been adapted to the modified loop structure. As a result, these transformations lead to different working spaces of the loops and thus change the locality behavior of the implementations. All general implementations have in common that the function evaluations of the right-hand-side function may access all components of the argument vector. Therefore, the parallelization of all general implementations is performed similarly to implementation (A) by inserting appropriate barriers at every stage to ensure that all components of the argument vector $\boldsymbol{w}$ are available before the function evaluation starts (Fig. 1).



(a) Cache line size 32 bytes (b) Cache line size 1024 bytes

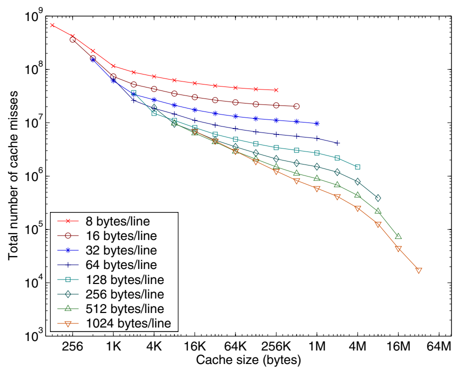
Fig. 2. Effect of different associativities on implementation (D) for BRUSS2D

Pipelining. The fact that for sparse ODE systems the function evaluations access only a small number of argument vector components within a bounded access distance enables us to perform a blockwise diagonal sweep across the stages rather than processing the stages successively [6]. Consequently, the working space of these implementations decreases to $\Theta(s^2b)$, where b is the blocksize (e.g., $b = 2N$ for BRUSS2D, $b \geq 2$ for MEDAKZO). In contrast, the working space of the general implementations consists of $\Theta(sn)$ vector components with $n = 2N^2$ for BRUSS2D and $n = 2N$ for MEDAKZO. The parallelization of the pipelining implementations can be performed using mutex variables to synchronize neighboring processors when they are finalizing their pipelines. But, using a modified data distribution, it is possible to reduce the synchronization overhead such that only one barrier is required at each time step.

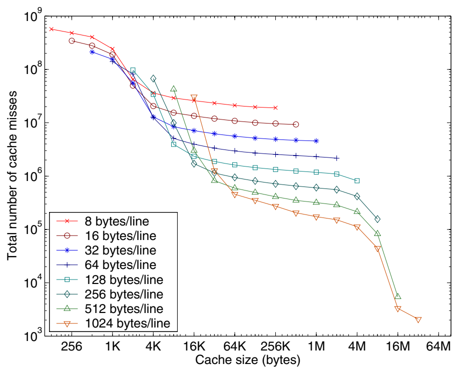
5 Experimental Results

To investigate the influence of different cache parameters on the locality behavior, we simulate a one-level unified cache and vary its parameters, i.e., the number of cache lines, the line size, and the associativity. In these experiments, starting at $t = 0$, we execute two time steps to warm-up the cache, and then measure the number of cache misses which occur during the execution of three further time steps. The results of the simulation are verified by runtime experiments on a real Sun Fire 6800 machine, where we measure the execution times of the implementations and investigate the cache behavior using hardware performance counters.

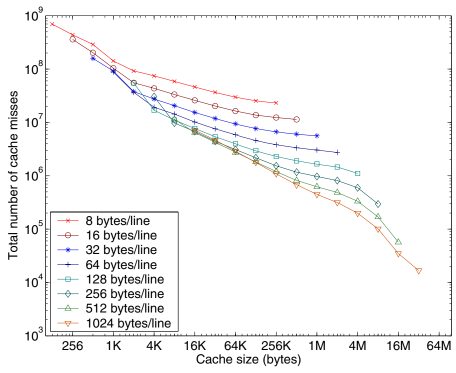
Influence of Associativity and Line Size. Figure 2 shows the number of cache misses of one general implementation as a function of cache size for different associativities for two different line sizes when solving the BRUSS2D problem. As expected, the number of cache misses of our implementations can be reduced by increasing the associativity of the cache. This particularly holds for caches with large cache lines. However, if the cache is sufficiently large (the exact size depends on the line size used) increasing



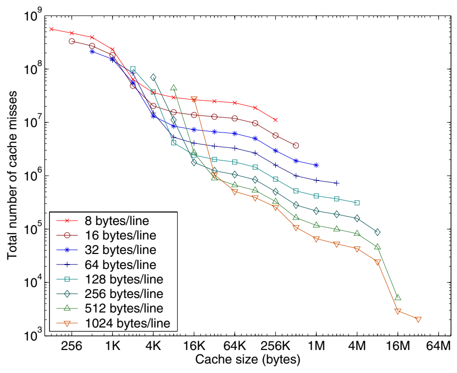
(a) Implementation (A)



(b) Implementation (D)



(c) Implementation (Dblock)



(d) Implementation (PipeD)

Fig. 3. Locality behavior for BRUSS2D for varying cache sizes and cache line sizes using one processor with a 16-way set-associative cache

the associativity to more than two ways improves the number of cache misses only insignificantly.

The influence of the cache line size can be observed from diagrams as those shown in Fig. 3, where for the different implementations the number of cache misses as a function of cache size is displayed for varying line sizes. All of our implementations profit from large line sizes. Actually, we observed the lowest number of cache misses for the largest line size we used (1024 bytes). The reason is that the implementations spend the main part of their execution time in processing vectors, i.e., consecutive memory regions, and can therefore take greater advantage of spatial locality if the line size is increased.

Working Sets. Figure 3 can also be used to identify the working sets [2] of the implementations, i.e., cache sizes where critical portions of an implementation's data set start fitting into the cache. The working sets correspond to the points of inflection of the curve representing the number of cache misses as a function of cache size and are best characterized by simulating a fully associative cache using a line size of one word.

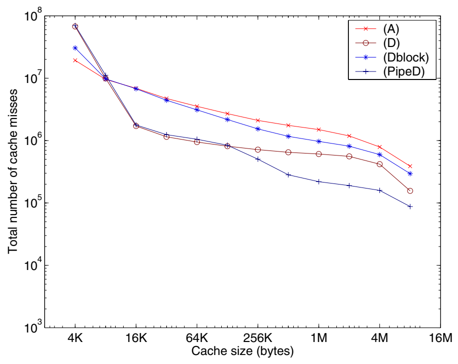
However, in our simulations we use a setup more close to practical systems, i.e., we use a 16-way set-associative cache, and line sizes between 8 bytes and 1024 bytes.

The most significant working set of all implementations is found between 8 and 16 MB for BRUSS2D and between 2 and 4 MB for MEDAKZO. This working set corresponds to the data accessed in one time step and is therefore independent of the loop structure used to compute the time steps. In addition, depending on the implementation, one or two other working sets can be identified. The first starts at about 1 to 4 KB. Its exact size depends on the problem, the implementation, and the cache line size used. This working set is characterized by a very smooth change of the number of cache misses resulting from the fact that the bounds of one loop, which occurs in all implementations, depend on the index of another loop. The pipelining implementations such as (PipeD) show another significant working set. Again, the exact size of this working set depends on the line size. For BRUSS2D, this working set starts between 128 KB and 512 KB. It results from the data accessed in the outer loop executed at every time step. The iterations of this loop, which runs over the elements of the ODE system, correspond to pipelining steps of the blockwise diagonal sweep across the stages.

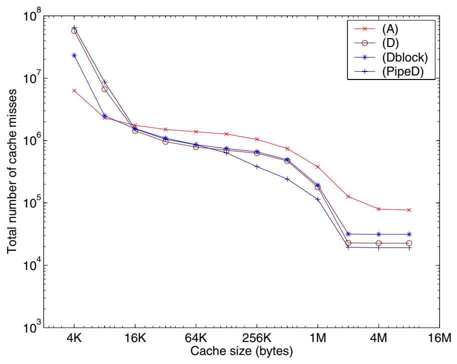
Comparison of the Implementations. Figure 4 shows a comparison of the number of cache misses of several embedded RK implementations. Comparing the two general implementations (A) and (D), we observe that (D) usually attains a lower number of cache misses than (A) if the cache size exceeds a certain size (e.g., 4–16 KB). The improved locality behavior of (D) is the result of several loop transformations (e.g., interchange and fusion) aiming at the frequent re-use of the results of the function evaluations [11]. These transformations replace the characteristic working set of (A) by a new working set which is slightly larger, but if it starts fitting into the cache, it decreases the number of cache misses further than the working set of the non-tightly nested loop structure of (A).

Our motivation to implement (Dblock) was to use *blocking* (also called *tiling*) to combine the good properties of (A) and (D). Our simulations show that this approach can lead to a good locality behavior for large as well as small caches. For example, the simulation of the sequential solution of MEDAKZO (Fig. 4(c)) and the simulation of the parallel solution of BRUSS2D (Fig. 4(b)) show a cache miss number close to (A) for cache sizes up to 16 KB where (A) leads to fewer cache misses than (D). If the cache size is larger than 16 KB, the number of cache misses of (Dblock) is close to (D) and significantly lower than the number of cache misses of (A). But implementation (Dblock) is not always successful in reducing the cache misses. For example, it cannot reach the low number of cache misses of (D) in the simulation of the sequential solution of BRUSS2D (Fig. 4(a)). In the simulation of the parallel solution of MEDAKZO, (Dblock) even generates a higher number of cache misses than (A) (Fig. 4(d)).

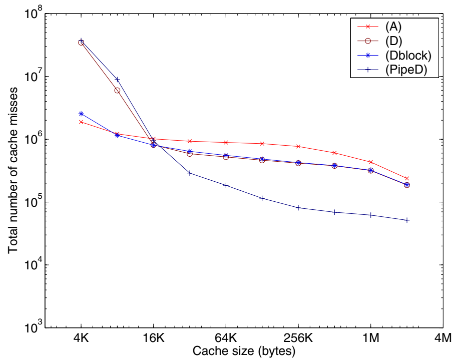
The pipelining implementation (PipeD) exploits the special access structure of the two problems by performing a blockwise diagonal sweep across the stages, i.e., the outer loop of the computational kernel of this implementation runs across the system dimension, and at every iteration of this loop (called a pipelining step) only a small part of the overall data set required to compute one time step is accessed. Therefore, this implementation shows a very low number of cache misses compared to the other



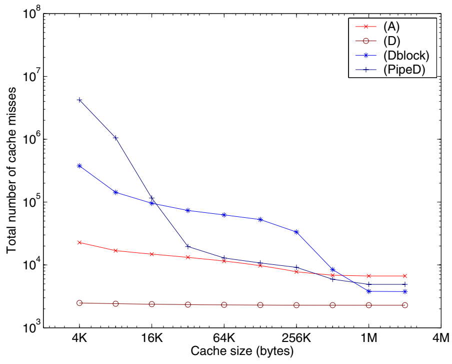
(a) BRUSS2D on one processor



(b) BRUSS2D on eight processors



(c) MEDAKZO on one processor



(d) MEDAKZO on eight processors

Fig. 4. Comparison of the locality behavior of different implementations on different numbers of processors (line size 256 bytes)

implementations in most of our experiments if the cache size is large enough to hold the working set of one pipelining step.

Real System. To verify the results of our simulations, we have performed experiments on a real Sun Fire 6800 where we investigate the runtimes (Tab. 1) and the locality behavior (Tab. 2). The cache hierarchy of the real Sun Fire consists of a 64 KB 4-way set-associative data cache (DC) and a 32 KB 4-way set associative instruction cache (IC) on the first level, and an 8 MB 2-way set associative unified ‘enhanced cache’ (EC) on the second level.

In these experiments, where we use the intergration intervals $[0, 0.5]$ for BRUSS2D and $[0, 10^{-4}]$ for MEDAKZO, the runtime is often closely related to the number of cache misses. Thus the order of the implementations w.r.t the runtime is often the same as the order w.r.t. the number of cache misses. But it is also very important how the functional units of the processors can be exploited. Therefore, (A) can outperform the other implementations in the experiments using MEDAKZO on one processor even though it shows a worse locality.

Since the large L2 cache can store a large part of the data accessed during one time step, the order of the implementations w.r.t. the number of cache misses is sometimes different compared with our simulations. E.g., using BRUSS2D with $N = 250$, implementation (A) attains a lower number of cache misses than (D) and (Dblock). But if we use $N = 750$, thus increasing the data set of one time step, (D) and (Dblock) show less cache misses than (A). (Dblock) usually attains at least a similar number of cache misses as (D), but often the number of cache misses is significantly lower than that of (D). In all our experiments, (Dblock) obtains a lower execution time than (D). (PipeD) shows the best locality behavior in most of our experiments. Therefore, only (A) can obtain a better runtime than (PipeD) in some experiments, because the compiler can exploit the simpler source code structure of (A) to produce more efficient machine code.

Table 1. Execution time (in seconds) on a real Sun Fire 6800

Problem	BRUSS2D				MEDAKZO	
	$N = 250$		$N = 750$		$N = 15000$	
Processors	1	8	1	8	1	8
(A)	4.65	0.61	651.3	48.4	134.0	25.9
(D)	4.92	0.63	530.5	49.3	143.2	24.5
(Dblock)	4.79	0.62	517.0	48.8	137.0	23.8
(PipeD)	4.74	0.62	382.6	45.4	136.5	21.3

Table 2. Number of cache misses (in thousands) on a real Sun Fire 6800

Problem	BRUSS2D				MEDAKZO	
	$N = 250$		$N = 750$		$N = 15000$	
Cache	DC	EC	DC	EC	DC	EC
(A)	981	785	53562	32062	120	98
(D)	1303	1156	26477	25013	69	76
(Dblock)	1192	1244	26542	20389	30	36
(PipeD)	254	650	4138	6424	21	85

6 Related Work

Loop transformations to improve the locality of scientific computations have been investigated by several authors. For example, [10] proposes computation regrouping to increase temporal locality, and [12] suggests the use of Cache Miss Equations (CMEs) to guide tiling and padding transformations. [7] gives an overview of different cache optimization techniques. There exist several alternative simulation environments suitable for investigating program locality, e.g., full-system simulators as SimpleScalar [1] and SimOS [5], and specialized cache simulators such as Dinero IV [3].

7 Conclusions

Simulation-based analysis enables us to investigate many interesting features of ODE solvers and other applications. Our experiments show that locality has a large influence on the execution speed of embedded RK implementations, particularly when executed in parallel. But other factors also influence the execution speed, such as synchronization and the utilization of functional units.

Which implementation performs best for a given problem depends on many factors, e.g., the system dimension, the access structure of the function evaluations, the particular memory hierarchy of the target system, and the optimizing capabilities of the compiler. If the access structure of the function evaluations enables the use of the pipelining computation scheme, it can be used to break down the working set of the outermost loop to fit into the cache. Otherwise, if a general implementation is required, we recommend using implementation (D) since its most significant working set reduces the number of cache misses further than the most significant working set of (A), and most modern microprocessors are equipped with L1 data caches of 64 KB or larger which usually can store the working set of (D). (Dblock) can run faster than (D) in many cases, but requires tuning of the blocksize.

References

1. T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *Computer*, 35(2):59–67, February 2002.
2. P. J. Denning. The working set model for program behavior. *Communications of the ACM*, 11(5):323–333, May 1968.
3. J. Edler and M. D. Hill. Dinero IV: Trace-driven uniprocessor cache simulator. `http://www.cs.wisc.edu/~markhill/DineroIV/`.
4. E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer, Berlin, 2nd edition, 2000.
5. S. A. Herrod. *Using Complete Machine Simulation to Understand Computer System Behavior*. PhD thesis, Stanford University, February 1998.
6. M. Korch and T. Rauber. Scalable parallel RK solvers for ODEs derived by the method of lines. In *Euro-Par 2003. Parallel Processing (LNCS 2790)*, pages 830–839. Springer, August 2003.
7. M. Kowarschik and C. Weiß. An overview of cache optimization techniques and cache-aware numerical algorithms. In *Proceedings of the GI-Dagstuhl Forschungsseminar: Algorithms for Memory Hierarchies*. Springer (LNCS 2625), 2003.
8. W. M. Lioen and J. J. B. de Swart. *Test Set for Initial Value Problem Solvers, Release 2.1*. CWI, Amsterdam, The Netherlands, September 1999.
9. P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, February 2002.
10. V. S. Pingali, S. A. McKee, W. C. Hsieh, and J. B. Carter. Restructuring computations for temporal data cache locality. *International Journal of Parallel Programming*, 31(4):306–338, August 2003.
11. T. Rauber and G. Rünger. Optimizing locality for ODE solvers. In *Proceedings of the 15th ACM International Conference on Supercomputing*, pages 123–132. ACM Press, 2001.
12. X. Vera, N. Bermudo, J. Llosa, and A. Gonzalez. A fast and accurate framework to analyze and optimize cache memory behavior. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 26(2):263–300, March 2004.
13. D. Wallin, H. Johansson, and S. Holmgren. Cache memory behavior of advanced PDE solvers. Technical Report 2003-044, Department of Information Technology, Uppsala University, August 2003.

A Novel Task Scheduling Algorithm for Distributed Heterogeneous Computing Systems

Guan-Joe Lai

Department of Computer and Information Science
National Tai-Chung Teachers College, Taichung, Taiwan, R.O.C.
gjlai@mail.ntctc.edu.tw

Abstract. This paper proposes a novel task scheduling algorithm to exploit the potential of parallel processing, allowing for system heterogeneity in distributed heterogeneous computing environments. Its goal is to achieve maximizing parallelization and minimizing communication. Due to that the algorithm avoids from the max-min anomaly in the parallelization problem and exploits schedule holes, it could produce better schedules than those obtained by existing algorithms. Experimental results are presented to verify the preceding claims. Three comparative algorithms are applied to demonstrate the proposed algorithm's effectiveness. As the system's heterogeneity increases, the performance improvement of the proposed algorithm becomes more outstanding than that of others. Therefore, the proposed scheduling algorithm may be used in designing efficient parallel environments for those situations where the system heterogeneity is the system performance bottleneck.

1 Introduction

This study proposes a novel task scheduling algorithm for distributed heterogeneous computing environments. A distributed heterogeneous computing system generally consists of a heterogeneous suite of machines, i.e., workstations or PCs. Although distributed heterogeneous computing systems offer significant advantages for high-performance computing, the effective use of heterogeneous resources still remains a major challenge. For effective utilization of diverse resources, an application could be partitioned into a set of tasks presented by an edge-weighted directed acyclic graph, such that each task could be scheduled to the best-suited machine to minimize the total completion time. Many heuristics have focused on solving the NP-complete problem [3] of efficiently scheduling tasks to heterogeneous computing systems, e.g., the generalized Dynamic Level Scheduling (DLS) algorithm [4], the Heterogeneous Earliest Finish Time (HEFT) [8] algorithm, the Critical Path on a Processor (CROP) [8] technique, the Iso-Level Heterogeneous Allocation (ILHA) algorithm [2], and the Partial Completion Time (PCT) algorithm [11]. However, most of these algorithms simply focus on computational aspects, so that the communication may become the system bottleneck as the computational power increases, particularly while executing applications with huge communication requirements. These strategies may perform poorly in distributed heterogeneous computing systems, owing to the heterogeneity of computational power and that of communication mechanisms.

A list-scheduling based algorithm, called the Dominant Tasks Scheduling (DTS) algorithm, is proposed here to exploit the potential of parallel processing, allowing for system heterogeneity and network bandwidth. In general, the list-scheduling approach assigns priorities to tasks and then allocates these tasks to machines in the order of their priorities to minimize a predefined cost function [18]. Most of them examine a ready task for scheduling only after all parents of the task have been scheduled. However, the operation of scheduling some ready tasks should not affect the strict reduction of the earliest starting time of the tasks in the critical path; otherwise, it may extend the overall parallel scheduling length. This situation is the so-called max-min anomaly in the parallelization problem [1]. Exploiting schedule holes [13] is another important issue in the scheduling problem. The schedule holes arise primarily because a task is scheduled after some other tasks with higher scheduling priorities, but could be scheduled before these tasks without affecting their earliest starting times. The DTS algorithm avoids from the max-min anomaly by taking account of the partial global scheduling information [15,16], and exploits schedule holes at each scheduling step. Therefore, it could produce better schedules than those obtained by existing algorithms.

2 Dominant Tasks Scheduling Algorithm

Before introducing the proposed algorithm, related terminology is presented. A parallel program is presented as a directed acyclic graph (DAG). The DAG is defined as G = (N, E, W, C), where N is the set of tasks, E is the set of communication edges, W is the set of task computation volumes, and C is the set of communication volumes. The value of $w_i \in W$ is the computation volume for $n_i \in N$. The value of $c_{ij} \in C$ is the communication volume occurring along $e_{ij} \in E$. Suppose that a distributed heterogeneous computing system is presented as M = (P, Q, A, B), where P is the set of heterogeneous processor elements, Q is the set of communication channels, A = $\{\alpha_i | i = 1, \ldots, |P|\}$ is the set of execution rates, and B = $\{\beta_{ij} | i = 1, \ldots, |P|, j = 1, \ldots, |P|\}$ is the set of transfer rates for the communication channels.

Let $p(n_i) \in P$ be the processor element to which n_i is allocated, $\alpha(p(n_i)) \in A$ be the execution rate for $p(n_i)$, and then $w_i \times \alpha(p(n_i))$ be the computation cost when the task n_i is allocated to $p(n_i)$. When n_i is allocated to $p(n_i)$ and n_j is allocated to $p(n_j)$, let $\beta(p(n_i), (p(n_j)) \in B$ be the transfer rate from $p(n_i)$ to $p(n_j)$, and then $c_{ij} \times \beta(p(n_i), (p(n_j))$ be the communication cost from n_i to n_j. Let $est(n_i)$ be the earliest starting time when n_i can start execution, and then the earliest completion time of n_i is defined as $ect(n_i) = est(n_i) + w_i \times \alpha(p(n_i))$. The least-completion time of $n_j \in succ(n_i)$, $lct(n_i) = max(c_{ij} \times \beta(p(n_i), (p(n_j)) + w_j \times \alpha(p(n_j)) + lct(n_j))$ is the least execution time from this node to the sink node, where $succ(n_i)$ is the set of immediate successors of n_i.

Given a DAG and a system as described above, the scheduling problem is to obtain the minimal-length, non-preemptive schedule of the task graph in the distributed heterogeneous computing systems. To prevent the max-min anomaly, the DTS algorithm reduces monotonically the intermediate scheduled length at each scheduling step to achieve a shorter final schedule length. The proposed algorithm identifies the dominant task in scheduling progress. A dominant task belongs in the critical path which dominates the partial scheduled length during the scheduling process. Therefore, the DTS algorithm could take advantage of the partial global information to monotonically reduce the scheduled length. In each scheduling step, the DTS algorithm finds out two ready candidates: the dominant task, n_{dt}, and the task, $n_{MaxPriority}$, with maximal scheduling priority. The dominant task is with the maximal value of

$$est(n_{dt}) + w(n_{dt}) \times \alpha(p(n_{dt})) + lct(n_{dt});$$

and, the scheduling priority function is $max(lct(n_{dt}) - est(n_{dt}) - w(n_{dt}) \times \alpha(p(n_{dt})))$. Then, the DTS algorithm schedules $n_{MaxPriority}$ to minimize $est(n_{MaxPriority})$ under the limitation of preventing $est(n_{dt})$ from being delayed.

In order to exploit schedule holes, the DTS algorithm also finds out the task, n_{sh}, which could be scheduled on $p(n_{MaxPriority})$ before $n_{MaxPriority}$, when the following conditions are satisfied: $ready(p(n_{MaxPriority})) \leq est(n_{sh})$, and $ect(n_{sh}) \leq est(n_{MaxPriority})$.

The DTS algorithm first finds the lct for each task, and then proceeds to deal with the set of ready tasks. When the set of ready tasks is not empty, the DTS finds out n_{dt} and $n_{MaxPriority}$. If n_{dt} is $n_{MaxPriority}$, then DTS sets n_{dt} to be the candidate. If n_{dt} is not $n_{MaxPriority}$, then DTS checks whether n_{dt} and $n_{MaxPriority}$ would be scheduled into the same processor element or not. If they are scheduled into the same processor element, then DTS tries to avoid from delaying the earliest starting time of n_{dt}. If they are not scheduled into the same processor element, then DTS sets $n_{MaxPriority}$ to be the candidate. When the candidate is found, DTS tries to find out the schedule hole before the candidate in the same processor element. If the schedule hole exists, then DTS sets n_{sh} to be the candidate. Finally, DTS schedules the candidate to its corresponding processor element, and updates the ready set. The DTS algorithm is listed as follows.

```
1.  Algorithm DTS
2.  Input: A system M=(P,Q,A,B); a DAG=(N,E,W,C).
3.  Output: A schedule with minimal parallel completion time.
4.  Finding $lct$ for each node.
5.  Finding the ready nodes.
6.  While the set of ready nodes is not empty
7.     Finding $n_{dt}$ and $n_{MaxPriority}$.
8.     If $n_{dt}$ = $n_{MaxPriority}$ then
9.        candidate = $n_{dt}$
10.    else
11.       If $p(n_{dt})$ = $p(n_{MaxPriority})$ then
12.          If scheduling $n_{MaxPriority}$ would delay $est(n_{dt})$ then
13.             Exiting while loop, and finding the next node with maximal
                scheduling priority.
14.          else
15.             candidate = $n_{MaxPriority}$.
16.          end if
17.       else
18.          candidate = $n_{MaxPriority}$.
19.       end if
```

```
20.     end if
21.     For candidate, finding the task, $n_{sh}$.
22.     If $n_{sh}$ exists then candidate = $n_{sh}$.
23.     Scheduling candidate to its corresponding processor element.
24.     Updating the set of ready nodes.
25.  end while
```

In order to provide constant measures of the scheduling performance, the performance bound of the DTS algorithm is analyzed to evaluate the accuracy of the heuristic solution. According to a previous theorem [6], the performance bound of the DTS scheduling algorithm ensures its performance within a factor of two times of the optimum for general directed acyclic task graphs, as shown in the following theorem.

Theorem 1. *For any DAG, G=(N, E, W, C) to be scheduled on a system M=(P, Q, A, B), the schedule length, ω, obtained by DTS always satisfies*

$$\omega \leq \left(2 - \frac{1}{|P|}\right)\omega_{opt} + c, \tag{2.1}$$

where ω_{opt} is the length of the optimal schedule, and c is a constant value.

Proof. The time in (0, ω) could be partitioned into two sets S_1 and S_2. S_1 is defined as the set of all points of times for which all processor elements are executing some tasks, and S_2 is defined as the set of all points of time for which at least one processor element is idle. If S_2 is empty, all processor elements complete their last assignment at ω and no idle interval can be found within (0, ω). The schedule is indeed optimal and, thus, the theorem holds. Therefore, we assume that S_2 is non-empty. Moreover, we also assume that S_2 is the disjoint union of q open intervals as below: S_2 = (I_{l1}, I_{r1}) $\cup$ (I_{l2}, I_{r2}) $\cup \ldots \cup$ (I_{lq}, I_{rq}), where $I_{l1} < I_{r1} < I_{l2} < I_{r2} < \ldots < I_{lq} < I_{rq}$. Without loss of generality, we claim that a chain of tasks (i.e., $X : n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow n_x$) could be found after applying DTS algorithm, such that the chain of tasks satisfies the following three cases.

(a) $st(n_x) \leq I_{l1}$.
(b) $st(n_x) \in S_2$, i.e., $\exists$ an integer h, $h \leq q$, s.t. $I_{lh} < st(n_x) < I_{rh}$.
(c) $st(n_x) \in S_1$ but $st(n_x) > I_{l1}$, i.e., $\exists$ an integer h, $h \leq q - 1$, s.t. $I_{rh} \leq st(n_x) \leq I_{l,h+1}$ or $I_{r,q} \leq st(n_x)$,

where n_x denotes the task that finishes in the DTS schedule at time ω, and $st(n_x)$ denotes the stating time scheduled by DTS for n_x.

If the task n_x is the entry task, the first possibility occurs; then the task n_x by itself constitutes a chain that satisfies our claim.

If the task n_x is not the entry task, the second and third possibilities may occur. In the scheduling process, there always is some task n_g, such that there exists a resource contention relationship with n_x; otherwise, DTS could advance the earliest starting time of the candidate, as shown in the above-described algorithm steps.

Then there always exists a h which satisfies (b) or (c). The cycle can be repeated by considering the above-mentioned three possibilities until the starting time of the last added task satisfies (a).

Finally, according to the theorem in [6], the performance bound of the DTS scheduling algorithm ensures its performance within a factor of two times of the optimum for general directed acyclic task graphs.

3 Experimental Results

Experimental results are presented to verify the preceding theoretical claims. Three algorithms (i.e., DLS [4], HEFT [8] and CROP [8]) are applied to comparatively demonstrate the proposed algorithm's effectiveness. The scheduling performance of these algorithms is compared in different communication/computation ratios (CCRs). In order to evaluate these algorithms, six practical parallel applications are applied, including the fork tree, the join tree, the fork-join, the FFT, the Gaussian elimination, and the LU decomposition. The experimental results show the superiority of the DTS algorithm.

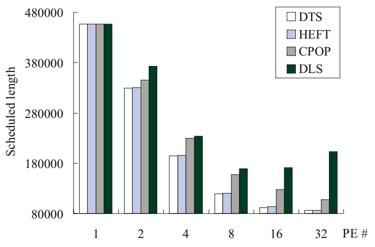


Fig. 1. Average scheduled lengths in six parallel applications

As the number of processor elements increases, the average scheduled lengths generated by the DTS, HEFT and CPOP decrease in the six parallel applications, as shown in Fig. 1. However, the average scheduled length generated by DLS does not decrease gradually, when the number of processor elements is larger than eight. It is due to that the DLS algorithm doesn't take the strictly reduction problem into consideration.

When the communication/computation ratio is set to 0.05, 0.1, or 1, the average scheduled lengths generated by the DTS, HEFT, CPOP and DLS in the six applications are respectively shown in Fig. 2. When CCR=0.05, the average scheduled lengths generated by the DTS and HEFT are similar, except that the DTS outperforms HEFT in terms of the schedule lengths in the FFT application, as shown in Fig. 2(a). When CCR=0.1, the average scheduled lengths generated by the DTS and HEFT are also similar; however, the HEFT outperforms the DTS in terms of the schedule lengths in the Fork-Join application, as shown in Fig. 2(b). As the communication/computation ratio increases to one, the DTS outperforms the HEFT in the bulk of applications, as shown in Fig. 2(c).

The variance of β indicates the heterogeneity of communication mechanisms, and the variance of α indicates the heterogeneity of computation power. As the value of

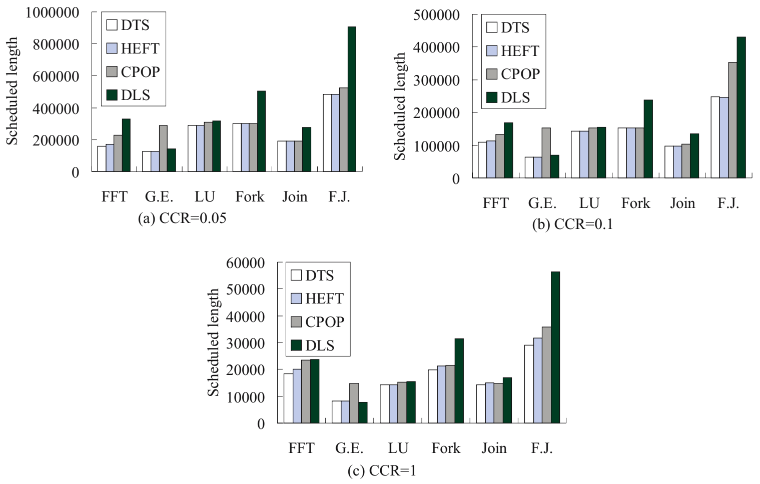


Fig. 2. Average scheduled lengths in different CCRs

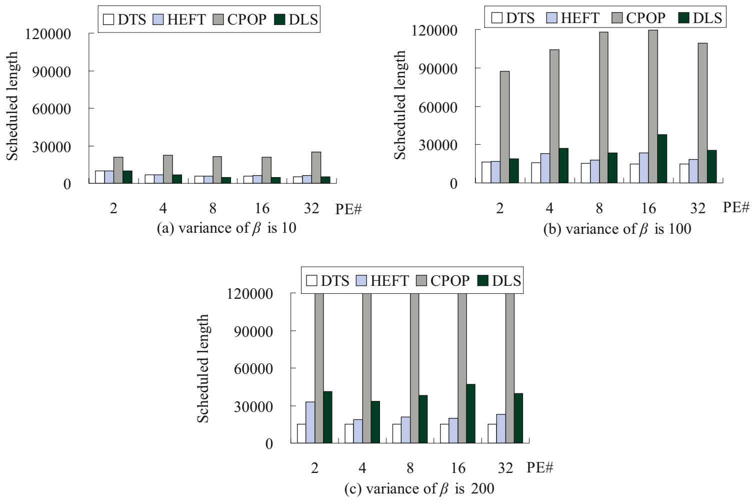


Fig. 3. Average scheduled lengths in different variances of β

the variance of β (or α) increases, the heterogeneity of communication mechanisms (or computation power) becomes obvious, and vice versa. In Fig. 3(a)-(c), the scales of the Y-axis are fixed from 0 to 120000. As the variance of β increases, the average scheduled lengths generated by four algorithms also increase, as shown in Fig. 3. Actually, we have shown the theoretical proof that the scheduling performance is affected by the heterogeneity of computational power and that of communication mechanisms in our previous work [5]; and Fig. 3 manifests it. [5] shows that the bounding margins of the scheduling performance are affected by the heterogeneity of computational power and

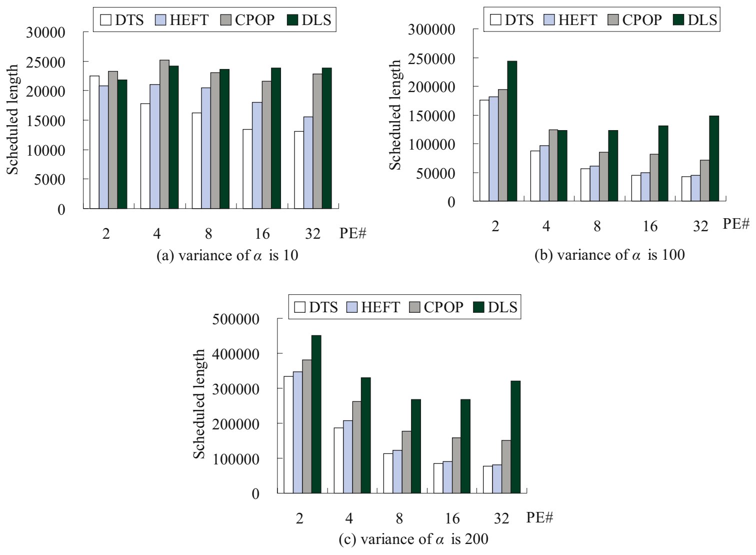


Fig. 4. Average scheduled lengths in different variances of α

that of communication mechanisms. As either heterogeneity increases, the spread of performance bounds also increases.

Because the scheduled length generated by DLS is over the upperbound of the Y-axis as the variance of β is 200, we ignore the exceeded part. From Fig. 3, we could observe that as the system heterogeneity increases, the performance improvement of the DTS algorithm becomes more stable than that of others. The similar situations could also be observed in Fig. 4.

4 Conclusions

In this paper, we present a novel task scheduling algorithm, DTS, for distributed heterogeneous computing systems. The DTS algorithm could take advantage of the partial global information to monotonically reduce the scheduled length, and to exploit schedule holes in each scheduling step. The performance of the DTS algorithm is demonstrated by evaluating six practical application benchmarks. Experimental results show the superiority of our proposed algorithm over those presented in previous literature, and that the scheduling performance is affected by the heterogeneity of computational power, the heterogeneity of communication mechanisms and the program structure of applications. As the system heterogeneity increases, the performance improvement of the DTS algorithm becomes more stable than that of others. Therefore, the proposed scheduling algorithm may be used in designing efficient parallel environments for those situations where the system heterogeneity is the system performance bottleneck.

Acknowledgements

This work was sponsored in part by the National Science Council of the Republic of China under the contract number: NSC92-2213-E-018-002.

References

1. B. Kruatrachue and T. G. Lewis. Grain Size Determination for Parallel Processing. *IEEE Software*, 23–32, 1988.
2. B. Olivier, B. Vincent and R. Yves. The Iso-Level Scheduling Heuristic for Heterogeneous Processors. *Proc. Of 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, 2002.
3. E. G. Coffman and P. J. Denning, Eds. *Operating Systems Theory*, Englewood Cliffs, NJ:Prentice-Hall, 1973.
4. G.C. Shih and E.A. Lee. A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. *IEEE Trans. on Parallel and Distributed Systems*, 4(2):175–187, 1993.
5. Guan-Joe Lai. Scheduling Communication-Aware Tasks on Distributed Heterogeneous Computing Systems. *International Journal of Embedded Systems*, accepted to be appeared, 2004.
6. Guan-Joe Lai. Performance Analysis of Communication-Aware Task Scheduling Algorithms for Heterogeneous Computing. *IEEE Proceedings of the 2003 Pacific Rim Conference on Communications, Computers and Signal Processing*, 788–791, August 2003.
7. H. El-Rewini and T. G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel Distribution Computing*, 9(2):138–153, June 1990.
8. H. Topcuoglu, S. Hariri, and M.Y. Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Trans. on Parallel and Distributed Systems*, 13(3):260–274, March 2002.
9. I. Ahmad and Y. Kwok. On Parallelizing the Multiprocessor Scheduling. *IEEE Trans. on Parallel and Distributed Systems*, 10(4):414–432, Apr. 1999.
10. M.A. Palis, J.-C. Liou, and D.S.L. Wei. Task Clustering and Scheduling for Distributed Memory Parallel Architectures. *IEEE Trans. on Parallel and Distributed Systems*, 7(1):46–55, Jan. 1996.
11. M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel Distribution Computing*, 59(2):107–121, Nov. 1999.
12. R. Andrei and J.C. Arjan. Low-Cost Task Scheduling for Distributed-Memory Machines. *IEEE Trans. on Parallel and Distributed Systems*, 13(6), June 2002.
13. S. Selvakumar and C. Siva Ram Murthy. Scheduling Precedence Constrained Task Graphs with Non-Negligible Intertask Communication onto Multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 5(3):328–336, 1994.
14. T. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hengsen, and R.F. Freund. A Comparison Study of Static Mapping Heuristics for a Classes of Meta-Tasks on Heterogeneous Computing Systems. *1999 Proc. Heterogeneous Computing Workshop*, 15–29, 1999.
15. T. Yang and A. Gerasoulis. DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors. *IEEE Trans. on Parallel and Distributed Systems*, 5(9):951–967, Sep. 1994.
16. Y. Kwok and I. Ahmad. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs onto Multi-Processors. *IEEE Trans. on Parallel and Distributed Systems*, 7(5):506–521, May 1996.
17. Y. Kwok and I. Ahmad. Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 31(4):406–471, Dec. 1999.
18. Y. Kwok and I. Ahmad. Static Scheduling Algorithms for Allocating Directed Task Graphs. *ACM Computing Surveys*, 31(4):406–471, Dec. 1999.

Study of Load Balancing Strategies for Finite Element Computations on Heterogeneous Clusters

Kalyani Munasinghe[1] and Richard Wait[2]

[1] Dept. of Computer Science, University of Ruhuna, Sri Lanka
kalyani@cc.ruh.ac.lk
[2] Dept. of Scientific Computing, Uppsala University, Sweden
richard@it.uu.se

Abstract. We study strategies for redistributing the load in an adaptive finite element computation performed on a cluster of workstations. The cluster is assumed to be a heterogeneous, multi-user computing environment. The performance of a particular processor depends on both static factors, such as the processor hardware and dynamic factors, such as the system load and the work of other users.
On a network, it is assumed that all processors are connected, but the topology of the finite element sub-domains can be interpreted as a processor topology and hence for each processor, it is possible to define set of neighbours. In finite element analysis, the quantity of computation on a processor is proportional to the size of the sub-domain plus some contribution from the neighbours. We consider schemes that modify the sub-domains by, in general, moving data to adjacent processors. The numerical experiments show the efficiency of the approach.

1 Introduction

The price-performance ratio of networks of workstations (NOWs) give better system architectures for parallel processing applications. Several projects have been initiated to deliver parallel supercomputing power by connecting large number of dedicated workstations. In addition, shared cluster networks of workstations also provide an attractive environment for parallel applications.

Increasing availability of low cost computers and networks has generated increased interest in the use of distributed systems as a parallel computational resource with which to solve large applications. Even though the demand for computing power is large, many existing systems are underutilized and a large amount of computing capacity remains unused. At any instant, on a particular network only a few servers may be working at capacity, most of the systems will not. Therefore many computing cycles that could be utilised to fulfill the increasing computational demand are unused. The search for ways to do more work with available resources by handling dynamically changing workloads is the main aim of our work.

Scheduling parallel applications on shared environments is different from scheduling in dedicated environments. The scheduling strategies of the dedicated environment depends on exclusively accessible identical processing elements. Scheduling in a shared environment is more challenging as the actual computing power available for parallel applications is dynamically changing. Some of the reasons are that the speed of machines are different, machines may fail and primary users generate workload which should be processed without interference by additional parallel computations.

In cluster environment, it is possible to get both high performance and high throughput if resources are allocated and managed efficiently. One of the problems in achieving high performance in clusters is that resources cannot be controlled by the individual program. Usually, networks, computers and disks are shared by among competing applications. When parallel programs execute in this type of an environment, these programs compete for resources with other programs and has to face resource fluctuation during execution. It is necessary to adapt these types of applications to changing system environment, in order to achieve high performance. As the distributed resources cannot be controlled by a single global scheduler, it is necessary to schedule the applications by the application developer. It is also necessary to take into account both application specific and dynamic system information in developing a schedule. Several factors such as type of resources allocated, machine performance can be considered. In this work, we examine how to adapt parallel applications in CPU sharing on workstation networks and clusters to achieve goals such as minimizing execution time.

For some irregular grid applications, the computational structure of the problem changes from one computational phase to another. For example, in an adaptive mesh, areas of the original graph are refined in order to model the problem accurately. This can lead to a highly localized load imbalance in subdomain sizes. Alternatively, load imbalance may arise due to variation of computational resources. For example in a shared network of workstations, computing power available for parallel applications is dynamically changing. The reasons may be that the speed of machines are different or there are other users on some part of the cluster, possibility with higher priority. The partitioning has to be altered to get a balanced load. We propose an algorithm which reduces the load imbalance by local adjustments of current loads to reduce the load spike as quickly as possible and to achieve a load balance. It is assumed that the connections for data transfers between the processors are determined by the data locality but data movement should be kept as low as possible. The load balance is adjusted in general by migrating data to adjacent processors with modifications to the connectivity where necessary.

On a homogeneous cluster of dedicated processors (e.g. Beowulf [3]) with a fixed problem size, the partition may be uniform and static.

2 Background

2.1 Some Definitions

Let p be the number of processors. The processor graph is represented by a graph (V, E) with $|V| = p$ vertices and $|E|$ edges. Two vertices i and j form an edge if processors i and j share a boundary of the partitioning. Hence the processor graph is defined by the topology of the data subdomains. As the edges of the processor graph are defined by the partitioning of the domain, when the partition changes the graph topology may change. Each vertex i is associated with a scalar l_i, which represents the load on the processor i.

The total load is

$$L = \sum_{i=1}^{p} l_i \tag{2.1}$$

The average load per processor is

$$\bar{l} = \frac{1}{p} L \tag{2.2}$$

and we can define the vector, b, of load imbalances as

$$b_i = l_i - \bar{l} \tag{2.3}$$

This definition is based on the assumption that in order to achieve a perfect balanced computation, all the loads should be equal. If however the processor environments are heterogeneous and corresponding to each processor there is a load index α_i which can be computed using current system and/or processor information, then the ideal load $\tilde{l}_i$ is defined as

$$\tilde{l}_i = \alpha_i \frac{1}{\sum_j \alpha_j} L \tag{2.4}$$

Load difference from the ideal load can be defined as

$$d_i = l_i - \tilde{l}_i \tag{2.5}$$

A processor is therefore overloaded if $d_i > 0$. These simple definitions assume that the computation can be broken down into a large number of small tasks each of which can be performed on any processor for the same computational cost. This is not necessarily true as for example in a finite element computation, the cost of the computation might depend on the number of edges between subdomains in addition to the cost proportional to the size of the subdomains. So the total distributed computational cost is not necessarily equal after any redistribution of the data.

In order to reduce any unnecessary data fragmentation, data will in general only be moved between contiguous subdomains. It is assumed that any processor is equally accessible from all other processors.

3 Existing Approaches

Some work [7] assume that the processors involved are continuously lightly loaded, but commonly the load on a workstation varies in an unpredictable manner. Weissman [11] worked on the problem adapting data parallel applications in a shared dynamic environment of workstation clusters. They have developed an analytical frame work to compare different adaptation strategies.

There are algorithms exist for scheduling parallel tasks. The Distributed Self Scheduling (DSS) [8] technique uses a combination of static and dynamic scheduling. During the initial static scheduling phase, *p* chunks of work are assigned to the *p* processors in the system. The first processor to finish executing its tasks from the static scheduling phase designates itself as the centralized processor and it stores the information about which tasks are yet to be executed, which processors are idle and dynamically distributes the tasks to the processors as they become idle.

Alessandro [4] introduced a method to obtain load balancing through data assignment on a heterogeneous cluster of workstations. This method is based on modified manager-workers model and achieves workload balancing by maximizing the useful CPU time for all the processes involved.

Dynamic load balancing scheme for distributed systems [6] considers the heterogeneity of processors by generating a relative performance weight for each processor. When distributing the workload among processors, the load is balanced proportional to these weights.

Many methods proposed in the literature to solve the load balancing problem are applicable to adaptive mesh computation. One of the earliest schemes was an iterative diffusion algorithm [5]. At each iteration, new load is calculated by combining the original load and the load of neighbouring processors. The advantage of this approach is, it requires local communication only, but the problem is its slow convergence. Horton [12] proposed a multilevel diffusion method by recursively bisecting a graph into two subgraphs and the balancing of the load of two subgraphs. This method assumes that the graph can be recursively bisected into two connected graphs. An alternative multilevel diffusion scheme [13] described. Walshaw [14] implemented a parallel partitioner and a direct diffusion partitioner that is based on the diffusion solver [15]. Several scratch-remap [9] and diffusion based [10] adaptive partitioning techniques have also been proposed.

These different approaches are suitable for different system environments and different computational environments. In our approach, we try to identify sharp increases of load and to reduce them quickly as possible without necessarily achieving a perfect load balance.

4 Repartitioning with Minimum Data Movement

In this section, we describe our proposed approach. It operates on the processor graph which describes the interconnection of the subdomains of a mesh that has been partitioned and distributed among the processors.

A processor is highly overloaded if the load difference is excessive

$$d_i > cl_i (\text{for some constant } c < 1)$$

and a partition is balanced if no processor is overloaded.

We assume that an overloaded node initiates the load balancing operation whenever it detects that it is overloaded. One of the important features of our approach is to capture the need for the processor load to adapt very quickly to external factors.

4.1 Proposed Approach

Define:

$\mathcal{W}$:={nodes with zero load}=$\emptyset$

Define:

$\mathcal{H}$={overloaded nodes}

$\mathcal{H}_0$={highly overloaded nodes}

– For each overloaded node, define its under loaded neighbours and a list of its under loaded neighbours of neighbours, so
  for each $m \in \mathcal{H}$, $\mathcal{N}_m$={under loaded neighbours of m}

The algorithm is explained in detail in the box below.

```
For each $m \in \mathcal{H}$
For each $k \in \mathcal{N}_m$
    $\mathcal{NN}_k$={under loaded neighbours of $k\}\backslash\mathcal{N}$
    If $\mathcal{NN}_k \neq \emptyset$
        $\mathcal{W}_k$={nodes to be removed}$\subset \{\mathcal{NN}_k \cup \{k\}\}$
        If $\mathcal{W}_k \neq \emptyset$
            Redistribute load from $\mathcal{W}_k$ among neighbours
            Modify $\mathcal{N}_m$ by removing nodes in $\mathcal{W}_k$ and adding neighbours
            as appropriate
            $\mathcal{W} := \mathcal{W} \cup \mathcal{W}_k$
        endif
    endif
endfor
For each $m \in \mathcal{H}$
    $\forall\, i \in \mathcal{N}_m$ move $l_i - \tilde{l}_i$ from $m$ to its neighbour $i$ by diffusion
endfor
For each $m \in \mathcal{H}_0$
    Assign set $\mathcal{W}_m \subset \mathcal{W}$ of empty nodes to $m$
    Assign load $\tilde{l}_i - l_i$ from node $m$ to $i \in \mathcal{W}_m$ by repartitioning remaining
    part of $l_m$ into $s + 1$ "unequal" parts.
endfor
```

5 Experimental Results

The experiments were performed on the problem of using the finite element method on an unstructured grid. Here we assumed that the computation is element based so that the load to be redistributed can be considered as repartitioning of the elements into subdomains. A test grid created using FEMLAB [2] was used in the experiments. The figure 1 shows the test grid used in the tests.

Our proposed algorithm was implemented on 8 *Sun* workstations connected by a 100 Mb/s Ethernet. All *Sun*s share a common file server and all files are equally accessible from each host due to the implemented Network File System. MPI was the communication library and *gcc* was the *C* compiler used. In this work, we used a simple load sensor which used *Unix* commands to collect the system information. The load sensor calculated the work load of each machine. Here we used a combination of processor speed and the work load of each machine as a load index (i.e. speed/work load). The figure 2 explains the speed of the machines and the figure 3 represents the work load of each machine collected by the load sensor and the figure 4 describes the load index used in the experiments.

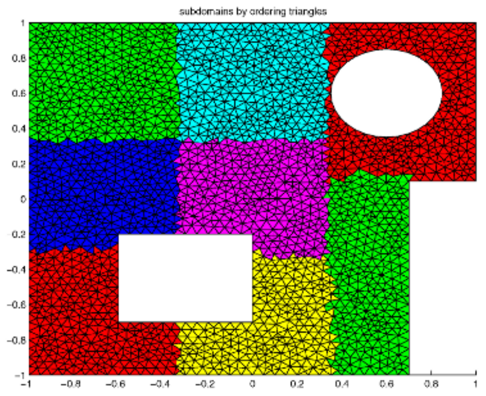


Fig. 1. Test Grid

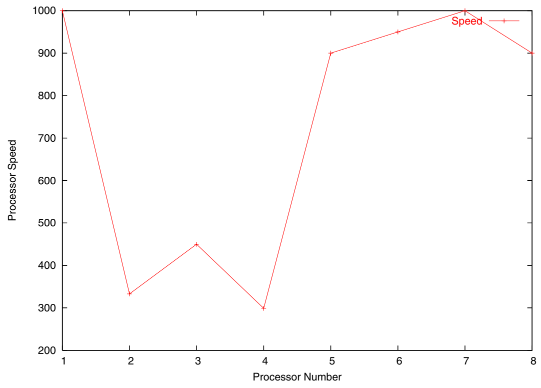


Fig. 2. Speed of processors

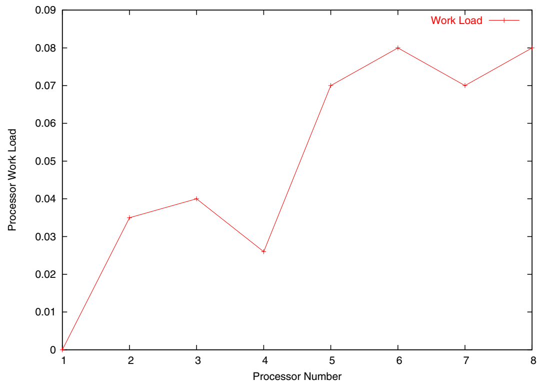


Fig. 3. Work load of processors

We ran the test without applying the load balancing approach and 3 iterations with the load balancing approach. The table 1 describes the advantage of the load balancing

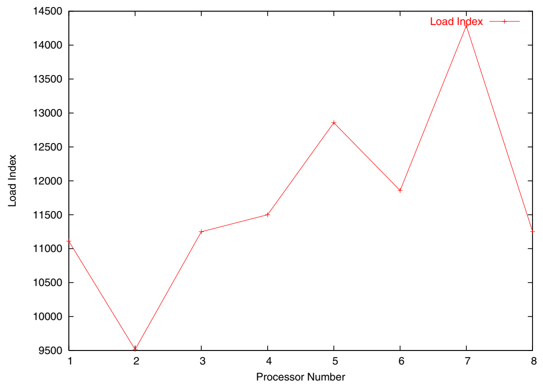


Fig. 4. Load Index for each processor

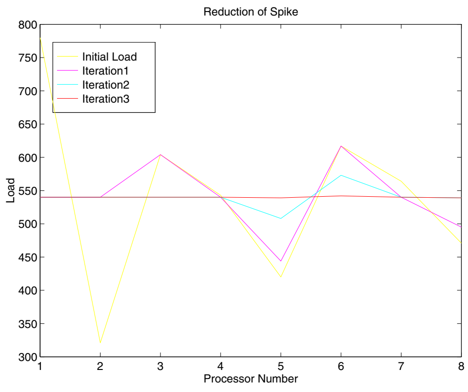


Fig. 5. Load Distribution in each Iteration

Table 1. Running Times in milliseconds

Without Load Balance	With Load Balance	
16.7276		
	iteration 1	2.41231
	iteration 2	2.21345
	iteration 3	2.16542

approach and the figure 5 shows how the computational work load is distributed in each iteration.

We also tested how the performance vary, when the load index varies. Table 2 gives a comparison of the timing.

Table 2. Different Load Index

Load Index	Run time in milliseconds
Processor speed x (1/work load)	2.41483
1/work load	5.6781
Available CPU	6.2142
Processor speed	11.2134

6 Conclusions

In this paper, we have presented an approach to reduce sharp increases of load as quickly as possible. The experimental results show a performance improvement with the approach. According to the experimental results, we can see that the load index also plays an important role. Our future work will focus on including more heterogeneous machines and large real data sets into our experiments.

References

1. http://www.-cse.ucsd.edu/users/breman/apples.html/.
2. http://www.comsol.com.
3. http://www.beowulf.org/.
4. Alessandro Bevilacqua, *A dynamic load balancing method on a heterogeneous cluster of workstations*, Informatica **23** (1999), no. 1, 49–56.
5. G. Cybenko, *Dynamic load balancing for distributed memory multiprocessors*, Parallel and Distributed Computing **7** (1989), 279–301.
6. Zhilling Lan and Valerie E. Taylor, *Dynamic load balancing of SAMR applications on distributed systems*, Scientific Programming **10** (2002), 319–328, no. 21.
7. C. K. Lee and M. Hamdi, *Parallel image processing application on a network of distributed workstations*, Parallel Computing **26** (1995), 137–160.
8. J. Lin and V. A. Saletore, *Self scheduling on distributed memory machines*, Supercomputing (1993), 814–823.
9. L. Oliker and R. Biswas, *Plum: Parallel load balancing for adaptive structured meshes*, Parallel and Distributed Computing **52** (1998), no. 2, 150–177.
10. Kirk Schloegel, George Karypis, and Vipin Kumar, *Multilevel diffusion schemes for repartitioning of adaptive meshes*, Journal of Parallel and Distributed Computing **47** (1997), no. 2, 109–124.
11. Jon B. Weissman, Predicting the Cost and Benefit of Adapting Data Parallel Applications in Clusters, Parallel and Distributed Computing (2002), 62, 1248–1271.
12. G. Horton, A multilevel diffusion method for dynamic load balancing, Parallel Computing (1993), 9, 209–218.
13. Kirk Schloegel and George Karypis and Vipin Kumar, Multilevel Diffusion Schemes for Repartitioning of Adaptive Meshes, Journal of Parallel and Distributed Computing (1997), 47, 2, 109–124.
14. C. Walshaw and M. Cross and M. G. Everett, Parallel Dynamic Graph Partitioning for Adaptive Unstructured Meshes, Journal of Parallel and Distributed Computing (1997), 47, 2, 102–108.
15. Y. F. Hu and R. J. Blake, An improved diffusion algorithm for dynamic load balancing, Parallel Computing (1999), 25, 4, 417–444.

Parallel Algorithms for the Determination of Lyapunov Characteristics of Large Nonlinear Dynamical Systems

Günter Radons[1], Gudula Rünger[2], Michael Schwind[2], and Hong-liu Yang[1]

[1] Institute of Physics, Technical University Chemnitz, 09111 Chemnitz, Germany
{radons,hya}@physik.tu-chemnitz.de

[2] Department of Computer Science, Technical University Chemnitz, 09111 Chemnitz, Germany
{ruenger,schwi}@informatik.tu-chemnitz.de

Abstract. Lyapunov vectors and exponents are of great importance for understanding the dynamics of many-particle systems. We present results of performance tests on different processor architectures of several parallel implementations for the calculation of all Lyapunov characteristics. For the most time consuming reorthogonalization steps, which have to be combined with molecular dynamics simulations, we tested different parallel versions of the Gram-Schmidt algorithm and of QR-decomposition. The latter gave the best results with respect to runtime and stability. For large systems the blockwise parallel Gram-Schmidt algorithm yields comparable runtime results.

1 Introduction

The Lyapunov characteristics of many-particle systems consisting of Lyapunov exponents and vectors found much interest recently [1,2]. Especially the dynamics of Lyapunov vectors which may show collective wave-like behavior, provides a challenge for current research [3,4,5,6,7,8,9,10,11]. In this paper we therefore consider means for effectively calculating the full spectrum of Lyapunov exponents and associated Lyapunov vectors for high-dimensional, continuous time nonlinear dynamical systems. The standard method of [12] is employed to calculate these quantities for our many-particle $(N = 100 - 1000)$ Lennard-Jones system in d dimensions. A system of $2dN \times 2dN$ linear and $2dN$ nonlinear ordinary differential equations has to be integrated simultaneously in order to obtain the dynamics of 2dN offset vectors in tangent space and the reference trajectory in phase space, respectively.

For the calculation of the Lyapunov exponents and vectors the offset vectors have to be reorthogonalized periodically using either Gram-Schmidt orthogonalization or QR decomposition. To obtain scientifically useful results one needs particle numbers in the above range and long integration times for the calculation of certain long time averages. This enforces the use of parallel implementations of the corresponding algorithms. It turns out that the repeated reorthogonalization is the most time consuming part of the algorithm. The algorithmic challenge for the parallelization lies on one hand on the parallel implementation of the reorthogonalization algorithm itself. On the other hand the results of the latter have to be fed back to the molecular dynamics integration routine, providing new initial conditions for the next reorthogonalization step, and so on. This alternation between reorthogonalization and molecular dynamics integration requires in addition an optimal balancing of computation and communication on distributed memory machines or cluster of SMPs.

As parallel reorthogonalization procedure we have realized and tested several parallel versions of Gram-Schmidt orthogonalization and of QR factorization based on blockwise Householder reflection. The parallel version of classical Gram-Schmidt (CGS) orthogonalization is enriched by a reorthogonalization test which avoids a loss of orthogonality by dynamically using iterated CGS. All parallel procedures are based on a 2-dimensional logical processor grid and a corresponding block-cyclic data distribution of the matrix of offset vectors. Row-cyclic and column-cyclic distributions are included due to parameterized block sizes, which can be chosen appropriately. Special care was also taken to offer a modular structure and the possibility for including efficient sequential basic operations, like from BLAS, in order to efficiently exploit the processor or node architecture. For comparison we consider the standard library routine for QR factorization from ScaLAPACK.

The interface between the parallel reorthogonalization and the integration procedure guarantees an invariant data distribution, which may require an automatic redistribution, so that different parallel orthogonalizations can be used within the integration process to adapt the parallel efficiency to the specific hardware properties. Performance tests are performed on a Beowulf cluster, a cluster of dual Xeon nodes, and an IBM Regatta p690+. Our emphasis is on a flexible combination of parallel reorthogonalization procedures and molecular dynamics integration that allows an easy restructuring of the parallel program. The goal is to exploit the characteristics of processors or nodes and of the interconnections network of the parallel hardware effectively by choosing an efficient combination of basic routines and parallel orthogonalization algorithm, so that computation of Lyapunov spectra and Lyapunov vectors can be performed in the most efficient way in order to be able to simulate very large problems.

The rest of the paper is organized as follows. Section 2 presents the physical problem of determining Lyapunov characteristics. Section 3 describes the parallel implementation of orthogonalization algorithms. Section 4 presents corresponding runtime results on different parallel machines and Section 5 concludes.

2 Determining Lyapunov Characteristics

The equations of motion for a many-body system may always be written as a set of first order differential equations $\dot{\Gamma}(t) = F(\Gamma(t))$ where Γ is a vector in the D-dimensional phase space. The tangent space dynamics describing infinitesimal perturbations around a reference trajectory $\Gamma(t)$ is given by

$$\delta\dot{\Gamma} = M(\Gamma(t)) \cdot \delta\Gamma \tag{2.1}$$

with the Jacobian $M = \frac{dF}{d\Gamma}$. The time averaged expansion or contraction rates of $\delta\Gamma(t)$ are given by the Lyapunov exponents. For a $D-$dimensional dynamical system there exist in total D Lyapunov exponents for D different directions in tangent space. The orientation vectors of these directions are the Lyapunov vectors $e^{(\alpha)}(t)$, $\alpha = 1,\cdots,D$.

In practice the Lyapunov vectors (LVs) $e^{(\alpha)}(t)$ and Lyapunov exponents $\lambda^{(\alpha)}$(LEs) are obtained by the so-called standard method consisting of repeated Gram-Schmidt orthogonalization of an evolving set of D perturbation vectors $\delta\Gamma(t)$ [12]. The time evolution of a matrix $E(t)$ composed of the D perturbation vectors is written as $E(t_{i+1}) = L \cdot E(t_i)$ where L is the propagator of $\delta\Gamma$. The computation of LEs and LVs can be presented as:

Algorithm 1 Computation of all the LEs and LVs of a dynamical system

```
Initialize $E_0$ as a $D \times D$ identity matrix
Initialize $LE\_vector$ as a zero $D$-dimensional vector
for $i = 1$ to $m\_iterations$
    $E_i = L_i \cdot E_{i-1}$                                   integration by MD simulation
    $E_i = Q_i \cdot R_i$                                      compute the QR factorization of $E_i$
    $LE\_vector = LE\_vector + log(diag(|R_i|))$
    for $j = 1$ to $D$
        $LV_j =j$-th row of $Q_i$
    end
    $E_i = Q_i$
end
$LE\_vector = LE\_vector/m\_iterations$
```

3 Parallel Realization

The entire parallel algorithm consists of integration steps which are periodically interrupted in order to apply a reorthogonalization of the offset matrix. Both parallel parts are based on the distribution of the matrix which differ according to efficiency needs. A natural data distribution in the integration parts is a column-block cyclic distribution. However in the orthogonalization part a different distribution may lead to the fastest algorithm so that a redistribution of data is needed when combining both parts. The redistribution of data is performed appropriately by an interface routine. The communication overhead of the redistribution can outbalance the advantage of a faster orthogonalization and a suboptimal orthogonalization can lead to the best parallel overall runtime.

3.1 Parallel Orthogonalization Algorithms

As described, orthogonalization plays an important role in the interplay with the molecular dynamics simulation. To efficiently perform reorthogonalization in such a context we have implemented and compared several parallel algorithms for the Gram-Schmidt-orthogonalization and for the QR-decomposition. The basis for the parallel algorithms is a two dimensional block cyclic data distribution and a logical two-dimensional processor grid. The parameters for the blocksize and the number of processors in each matrix dimension can be chosen according to the needs for the algorithms. For the use within the program for the determination of Lyapunov characteristics the following three parallel Gram-Schmidt versions and two QR decompositions have been implemented.

- The **Parallel-Classical-Gram-Schmidt-Algorithm (PCGS)** is a parallel implementation of the Classical-Gram-Schmidt-Algorithm (CGS) [13]. Due to the structure of the Gram-Schmidt algorithm and the distribution of the matrix the R-Matrix is available in transposed form which is distributed on the processor grid. The PCGS-Algorithm may have the disadvantage that the calculated Q-Matrix is not always orthogonal for matrices with high condition numbers [14].
- The **Iterated-Parallel-Classical-Gram-Schmidt-Algorithm (PICGS)** has a similar structure as PCGS but includes an iterative reorthogonalization which avoids the disadvantage of PCGS. The need for a repeated orthogonalization is determined by a test described in [15].
- The **Parallel-Blockwise-Classical-Gram-Schmidt-Algorithm (PBCGS)** is a block version of the CGS-Algorithm (Algorithm 2.4 in [16]) which uses highly optimized cache efficient matrix-matrix operations (BLAS [17,18] Level 3 kernels).
- The **Parallel-QR-Algorithm (PQR)** is a parallel implementation of the QR-Algorithm with Householder-Reflectors. The parallel implementation consists of two steps. First the R-Matrix and the Householder-Reflectors are calculated. In the second step the Q-Matrix is constructed.
- The **Parallel-Blockwise-QR-Algorithm (PBQR)** is a parallel QR-Algorithm with Householder Reflectors [19,20,21] which computes column blocks instead of single columns. Like in the PQR-Algorithm the R-Matrix is calculated before the matrix Q is calculated in the second step. For the blocking the WY-Representation for products of Householder-Vectors in [21] is used, which allows the use of cache efficient matrix-matrix operations. A description of the optimizations of the algorithm is given in Section 3.3.

The parallel algorithms are mainly derived from the sequential algorithms by substituting the building blocks (vector-, matrix-vector-, or matrix-matrix operations) by parallel equivalents which work over the block cyclic-distribution on the logical processor grid. All parallel programs are designed in an SPMD programming style and use the Message-Passing-Interface (MPI) Standard for communication [23]. The Gram-Schmidt algorithms construct the R-Matrix column by column. The decision to apply a second orthogonalization step in the PICGS-Algorithm is done in one processor column of the processor grid and then it is broadcasted within the processor rows. The difference between our algorithm and the ScaLAPACK-Routine `PDGEQRF` lies in the fact that our algorithm caches some calculations in the first stage to use them in the second, which is not done in ScaLAPACK [24].

3.2 Interface to Orthogonalization Routines

To include the parallel orthogonalization modules into the entire program we have developed a generic interface routine which calls the different orthogonalization algorithms. The routine is able to redistribute the input matrix to a data distribution format needed by the integration part of the program. The data distribution format differs in the lengths of blocksizes and the number of processors in each matrix dimension. The distribution in the orthogonalization part is adapted to use the underlying hardware platform as efficiently as possible. As input data this routine requires the matrix to be orthogonalized and a number of parameters which describe the distribution of the input matrix and the distribution needed for the calculation. This routine outputs the orthogonalized matrix and on request the R-Matrix or the diagonal elements of the R-Matrix.

3.3 Optimizations for the PBQR-Algorithm

As mentioned above the PBQR-Algorithm uses the WY-Representation for products of Householder Matrices described in [21], which has the form $I - VTV^T$. V is a trapezoidal matrix with 1 in its diagonal. T is an upper triangular matrix. The block representation is used both for computing the R- and Q-matrix. Thus in contrast to ScaLAPACK implementation we avoid a second computation of the T-matrix and store the T-Matrix for the computation of the Q-matrix.

A straightforward parallelization of the Block-QR-Algorithm (like PBQR) with distributed BLAS-Operations (e.g. PBLAS) would result in a load imbalance because of idle times for some processors. When one column of the two dimensional processor grid computes the block of Householder-Reflectors with Level 2 Blas operations, the processors in the other processor-columns wait for the result of this computation before they can update the remaining matrix.

As optimization the computation we have realized a more flexible computation scheme on the logical processor grid which makes an overlapping of communication and computation possible. A similar scheme has been presented in [25] for a ring of processors. The more flexible implementation requires a non-blocking broadcast operation, which is not available in the MPI-Standard. While the JUMP-architecture provides an MPE_Ibcast, we have implemented non-blocking broadcast operations on the two other architectures by using nonblocking send-receives.

4 Performance Results

The left column of Figure 1 shows the runtime plots for the entire molecular dynamics simulation on the three architectures listed in Table 1. The PBQR algorithm outperforms the Gram-Schmidt versions on the CLIC and XEON cluster and has a similar runtime on the JUMP architecture. The moderate increase in runtime of the PICGS indicates a good condition of the matrix of offset vectors. The PQR-algorithm gives the best performance on the CLIC-architecture for the column-block-cyclic distribution which is used by the integration-part.

The speedup of the program is mainly limited by the orthogonalization routine which can be seen from the right column of Figure 1. Here the percentage of runtime used by the orthogonalization part increases with higher numbers of processors. Due to the fact that the molecular dynamics part scales fine with higher numbers of processors the total runtime decreases even if the runtime of the orthogonalization algorithm does not decrease or even increases slightly.

Test runs with redistribution show that the overhead of redistributing the input- and output-matrices limits the advantage of the optimal processor grid for small matrices and small numbers of processor. Figure 2 shows a comparison of parallel runtimes for distributions on different processor grids: a column-block-cyclic distribution where all

Table 1. Systems used for runtime measurements

Name	Processor	Communication	BLAS
CLIC	Pentium III @800MHz	LAM-MPI over Ethernet	libgoto for PIII [22]
XEON	dual Xeon @2GHz	Scali over SCI	libgoto for PIV [22]
JUMP	32x Power4+ @1.7GHz	1 Node Shmem MPI	essl

processors form one row of the form $(1 \times P)$, a row-block-cyclic distribution where all processors are in one column of the form $(P \times 1)$, and a block-cyclic distribution in both dimension of size $P_1 \times P_1 = P$. The distributions $1 \times P$ and $P \times 1$ are not optimal and are outperformed by the square processor grid layout.

An analysis of our algorithms in terms of block sizes for the block-cyclic distribution demonstrates that the PCGS-algorithm has a minimum runtime for a column blocksize of one. The blocked algorithms PBCGS and PBQR show a behavior similar to that of Figure 3 which shows the runtimes on the XEON-Cluster for a 1200×1200 matrix on a 4×4 processor-grid. This figure shows a high runtime increase for small column blocksizes. The runtime of these two algorithms is not very sensitive with respect to a change of the row blocksize. The optimal column-blocksize for the block-algorithms decreases with an increasing number of processor-grid-columns when the number of the processors in the row is held constant.

Figure 4 compares the runtime of the different QR-Algorithms with Householder-Reflectors with the ScaLAPACK-Implementation consisting of one call to PDGEQRF (computing the R-Matrix) and PDORGQR (forming the Q-matrix). The measurement was done with 16 processors in the grid configurations (1×16), (2×8), (4×4), (8×2) and (16×1). In Figure 4 only the best runtime measured for a specific matrix dimension is plotted. The Figure 4 shows that our algorithms are comparable in runtime on the CLIC-Architecture and outperform the ScaLAPACK implementation on the JUMP- and XEON-Clusters by a maximum of up to 35 and 46 percent. The best runtimes have been measured with the PBQR with the optimization described in Subsection 3.3 (PBQR2 in Figure 4).

5 Conclusion

We have developed and tested several parallel implementations for the calculation of all Lyapunov characteristics, vectors and exponents, of large dynamical systems. The latter are of great importance for an understanding of the dynamics of many-particle systems. It turned out that in the interplay between molecular dynamics (MD) integration and repeated reorthogonalization the latter is the critical factor, which for large systems delimits overall performance. Among the five tested reorthogonalization algorithms the Parallel-Blockwise -QR Algorithm based on Householder reflectors (PBQR) gave the best results with respect to runtime and stability. They also use most effectively the column-cyclic data distribution generated naturally by the MD-algorithm. For large systems the blockwise parallel Gram-Schmidt algorithm yields comparable results. With such an implementation the investigation of Lyapunov instabilities in 2- and 3-dimensional systems with several hundred particles becomes possible.

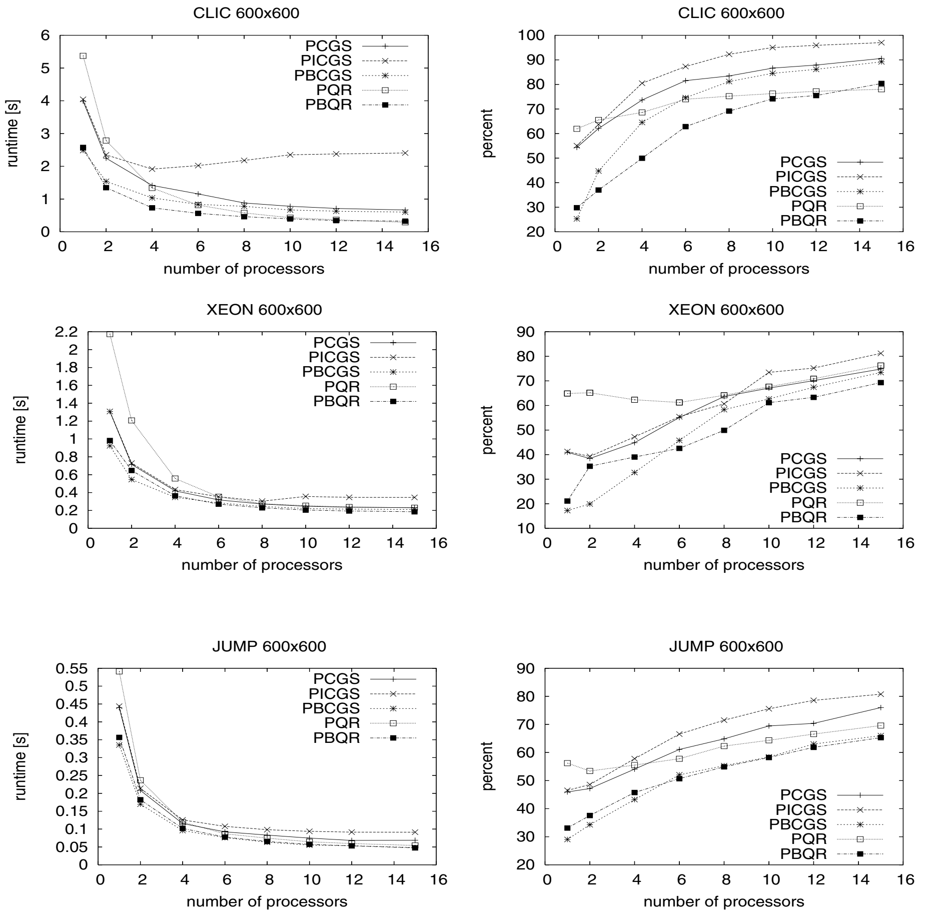
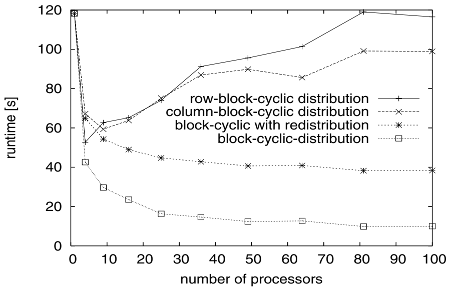


Fig. 1. Parallel runtime per iteration step of the molecular dynamics simulation including reorthogonalization (left column) for the CLIC, XEON, and JUMP systems (from the top). The right column shows the corresponding percentages of runtime used for the orthogonalization part alone. The measurements are made on the architectures listed in Table 1. The experiments have been performed with a 600×600 matrix of offset vectors per run. The orthogonalization algorithm works without redistribution

Acknowledgment

We thank the National Supercomputer Centre *John von Neumann-Institute for Computing (NIC)* in Jülich, Germany, for providing access to the Jump Cluster. We gratefully acknowledge financial support from the DFG within Sonderforschungsbereich 393 *Parallele Numerische Simulation für Physik und Kontinuumsmechanik*.



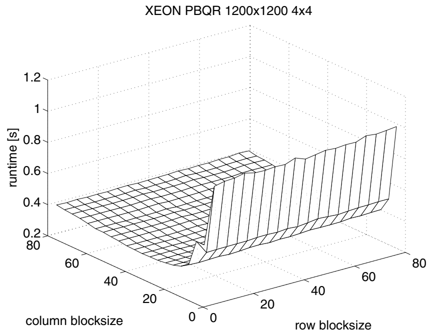


Fig. 2. Runtime for a single orthogonalization for the PBCGS-algorithm on CLIC for a 3000×3000 matrix

Fig. 3. Runtime on the XEON-Cluster for a 4×4 processor grid with a total of 16 processors for the PBQR-algorithm

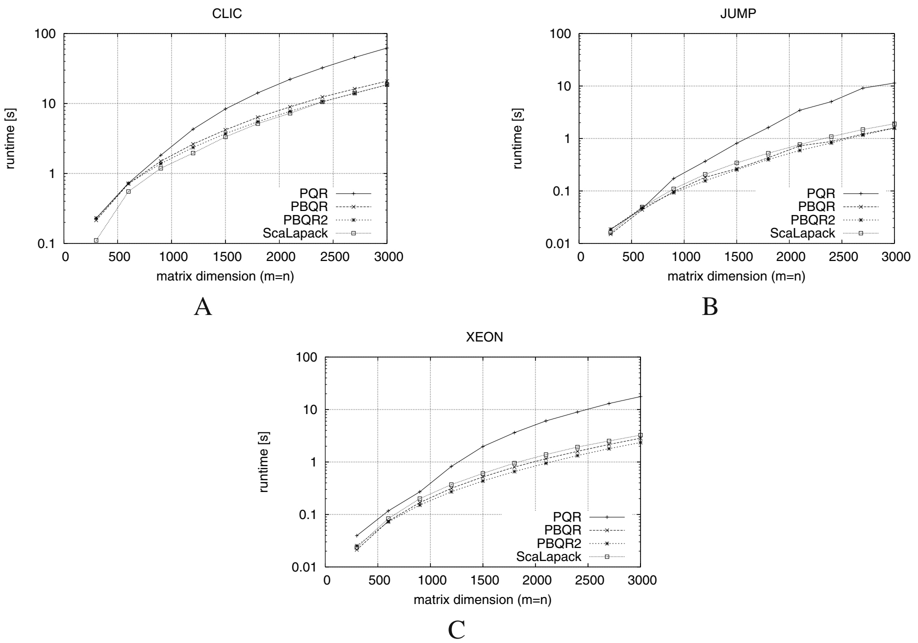


Fig. 4. Parallel runtime of the QR-Algorithms with Householder-Reflectors in isolation without the simulation- and redistribution-part for the block-cyclic distribution shown in dependency of the matrix dimension. The measurement was done with random square-matrices on 16 processors. The ScaLAPACK algorithm consists of a call to PDGEQRF and PDORGQR

References

1. P. Gaspard, *Chaos, Scattering, and Statistical Mechanics* (Cambridge University Press, Cambridge, 1998).
2. J.P. Dorfman, *An Introduction to Chaos in Nonequilibrium Statistical Mechanics* (Cambridge University Press, Cambridge, 1999).
3. H.A. Posch and R. Hirschl, "Simulation of Billiards and of Hard-Body Fluids", in *Hard Ball Systems and the Lorenz Gas*, edited by D. Szasz, Encyclopedia of the mathematical sciences **101**, p. 269, (Springer, Berlin, 2000).
4. C. Forster, R. Hirschl, H.A. Posch and Wm.G. Hoover, Physics D **187**, 294 (2004).
5. H.A. Posch and Ch. Forster, Lyapunov Instability of Fluids, in *Collective Dynamics of Nonlinear and Disordered Systems*, p.301, Eds. G. Radons, W. Just, and P. Häussler (Springer, Berlin, 2004).
6. J.-P. Eckmann and O. Gat, J. Stat. Phys **98**, 775 (2000).
7. S. McNamara and M. Mareschal, Phys. Rev. E **64**, 051103 (2001). M. Mareschal and S. McNamara, Physics D **187**, 311 (2004).
8. A. de Wijn and H. van Beijeren, Goldstone modes in Lyapunov spectra of hard sphere systems, nlin.CD/0312051.
9. T. Taniguchi and G.P. Morriss, Phys. Rev. E **65**, 056202 (2002); ibid, **68**, 026218 (2003).
10. H. L. Yang and G. Radons, Lyapunov instability of Lennard Jones fluids, nlin.CD/0404027.
11. G. Radons and H. L. Yang, Static and Dynamic Correlations in Many-Particle Lyapunov Vectors, nlin.CD/0404028.
12. G. Benettin, L. Galgani and J. M. Strelcyn, *Phys. Rev. A* **14**, 2338 (1976).
13. J. Stoer and R. Bulirsch, *Introduction to Numerical Analysis*, Springer, 2002.
14. Å. Björck, Numerics of Gram-Schmidt Orthogonalization, *Linear Algebra Appl.*, 197–198:297–316, 1994.
15. Luc Girand, Julien Langou and Miroslav Rozlǒzník, On the round-off error analysis of the Gram-Schmidt algorithm with reorthogonalization, www.cerfacs.fr/algor/reports/2002/TR_PA_02_33.pdf.
16. D. Vanderstraeten, An accurate parallel block Gram-Schmidt algorithm without reorthogonalization, *Numer. Linear Algebra Appl.*, 7(4):219–236, 2000.
17. J. Dongarra, J. Du Croz, I. Duff, S. Hammarling and Richard J. Hanson, An Extended Set of Fortran Basic Linear Algebra Subroutines, *ACM Trans. Math. Soft.*, 14 (1):1-17, March 1988.
18. J. Dongarra, J. Du Croz, I. Duff and S. Hammarling, A set of Level 3 Basic Linear Algebra Subprograms, *ACM Trans. Math. Soft.*, 16(1):1-17, March 1990.
19. Gene H. Golub and Charles F. Van Loan, Matrix Computations, Johns Hopkins University Press, 1996.
20. C. H. Bischof and C. Van Loan, The WY Representation for Products of Householder Matrices, *SIAM Journal on Scientific and Statistical Computing*, 8:s2–s13, 1987.
21. Schreiber and C. Van Loan, A Storage Efficient WY Representation for Products of Householder Transformations, *SIAM Journal on Scientific and Statistical Computing*, 10:53–57, 1989.
22. K. Goto and R. van de Geijn, On Reducing TLB Misses in Matrix Multiplication, FLAME Working Note #9, The University of Texas at Austin, Department of Computer Sciences, Technical Report TR-2002-55, Nov. 2002.
23. Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, Technical Report UT-CS-94-230, 1994.

24. L. S. Blackford, et al., ScaLAPACK Users' Guide, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.
25. K. Dackland, E. Elmroth, and B. Kågström, A Ring-Oriented Approach for Block Matrix Factorizations on Shared and Distributed Memory Architectures, *Proceedings of the Sixth SIAM Conference on Parallel Processing for Scientific Computing*, 330–338, Norfolk, 1993. SIAM Publications.

Online Task Scheduling on Heterogeneous Clusters: An Experimental Study

Einar M.R. Rosenvinge, Anne C. Elster, and Cyril Banino

Norwegian University of Science and Technology (NTNU)
Department of Computer and Information Science (IDI)
einarmr@tihlde.org, {elster,banino}@idi.ntnu.no

Abstract. This paper considers effcient task scheduling methods for applications on heterogeneous clusters. The Master/Worker paradigm is used, where the independent tasks are maintained by a master node which hands out batches of a variable amount of tasks to requesting worker nodes. The Monitor strategy is introduced and compared to other strategies suggested in the literature. Our online strategy is especially suitable for heterogeneous clusters with dynamic loads.

1 Introduction

In today's international high-performance computing arena, there is a clear trend from traditional supercomputers towards cluster and Grid computing solutions. This is mainly motivated by the fact that clusters typically can be constructed at a cost that is modest compared to the cost for traditional supercomputers that have equivalent computing power. The operation, use and performance characteristics of such clusters are, however, significantly different from those of traditional supercomputers. For instance, clusters typically have a much slower communication medium between nodes (e.g. a high-latency, low-bandwidth interface such as Ethernet or the faster Myrinet.)

Clusters also give rise to some challenges not typically found on traditional supercomputers. A cluster may be a *heterogeneous* environment, meaning that its nodes may have different performance characteristics. Also, if the nodes composing a cluster are not dedicated, the cluster will be a *dynamic* environment, because its nodes may have a non-negligible background processing load. These challenges imply that one needs an adequate scheduling strategy to get good performance on a cluster.

Our work aims at developing effective scheduling strategies for clusters for the class of applications that fall into the Master/Worker paradigm. These applications can be divided into a large number of independent *work units*, or *tasks*. There is no inter-task communication, so the tasks can be computed in any order. Finally, the tasks are atomic, i.e. their computation cannot be preempted. Many applications can be parallelized in such a way, including matrix multiplication, Gaussian elimination, image processing applications such as ray-tracing [1] and Monte Carlo simulations [2].

The rest of this paper is organized as follows: The test-case application and the test-bed cluster are described in Section 2. In Section 3, previous scheduling strategies, as well as our scheduling strategy, are presented. Section 4 exposes implementation-specific issues, and Section 5 discusses empirical results from our work. Finally, conclusions and suggestions for future work are provided in Section 6.

2 Framework

2.1 Test-Case Application: Matched Filtering

The test-case application used in this study is an image filtering application, known as *matched filtering* [3]. This application has been developed by O. C. Eidheim, a PhD student at IDI/NTNU, who was looking for speed improvements, giving us great access to a developer of the application.

The matched filtering application is used in medical imaging in order to detect blood vessels in Computer Tomography (CT) images. A CT image is a cross-section of the human body. By using this application to detect blood vessels in multiple adjacent CT images, one is able to construct a 3D representation of the blood vessels in the human body.

The input to the application is a grayscale image, see Fig. 1 (a), which is filtered through an image correlation step. The correlation kernel that is used is a Gaussian hill, which is rotated in all directions and scaled to several sizes. For more detailed information about this filtering technique, see [3]. The noise in the input image, makes the blood vessel identification quite challenging. After filtering, see Fig. 1 (b), the noise has been removed and blood vessels are now identifiable.

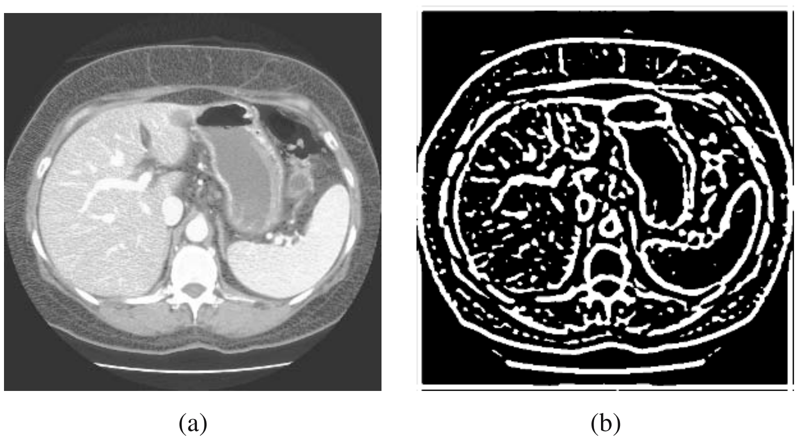


Fig. 1. Image before filtering (a), and after filtering (b). CT image courtesy of Interventional Center, Rikshospitalet, Oslo, Norway

Since the input image can be divided into tasks corresponding to different parts (lines, columns, blocks), each node can process one or more tasks, and thus produce the corresponding parts of the output image, this application parallelizes easily in a homogeneous and static environment. However, in a heterogeneous and/or dynamic environment provided by most of today's clusters, parallelizing this application efficiently is more complicated.

2.2 Test-Bed Platform: Athlon-Based Cluster

ClustIS is a fairly homogeneous cluster composed of 38 nodes, with AMD Athlon XP/MP CPUs at clock frequencies of 1.4 to 1.66 GHz with 0.5 to 2 GB of RAM. A few of the nodes are dual-CPU nodes. The nodes are connected through 100Mbit switched Ethernet. The operating system is Linux, the MPI implementation is MPICH 1.2.5.2, and the queuing system is OpenPBS.

On ClustIS, data storage is provided by one node, *Story*, which provides home directories through NFS. Consequently, all disk I/O from the nodes will go through this slow Ethernet interface. One solution could be to use local disk I/O instead. However, the scattering of input data and gathering of output data would add to the total application execution time, so regardless, input and output data would have to travel through the network.

Nevertheless, we were able to demonstrate some I/O parallelism on this cluster. In fact, we got more or less linear speedup when reading data concurrently from up to 8 processes. This indicates that having the worker nodes read their part of the data themselves will be faster than having the master scatter and gather data to/from workers. Writing data in parallel also gave a significant speedup compared to centralized writing, but the speedup was not quite as linear. See [4] for details.

3 Scheduling Master/Worker Applications

On a cluster, each processor might have very different performance characteristics (heterogeneity), as well as varying background workloads (dynamism). To a certain degree, heterogeneity can be handled through the job scheduler, by requesting processors with a certain CPU frequency. Such functionality, however, is not available with many job scheduling systems.

Dynamism, however, cannot be handled through a job scheduler. The background processing load of the processors is unknown before the computation starts, and might vary throughout the computation. This must therefore be handled by the scheduling strategy used.

3.1 Previous Scheduling Strategies

All popular scheduling strategies give out *batches* of tasks to workers, but since the workers might have different and possibly varying processing speeds, giving only one batch to each worker might lead to non-equal finish times for the workers. To compensate for this, some strategies give batches to workers in several *rounds*. In the following, N denotes the total number of tasks, p denotes the number of workers (processors), and R denotes the number of remaining unassigned tasks on the master at a given time.

The *Static Chunking (SC)* strategy [1] assigns one batch of N/p tasks to each worker. At the other end of the spectrum is the *Self Scheduling (SS)* strategy [1], where tasks are handed out one by one. The *Fixed-Size Chunking (FSC)* strategy uses batches of tasks of one fixed size, and it is possible to approximate the optimal batch size [5]. The *Guided Self Scheduling (GSS)* strategy [6] gives each worker batches of size R/p. GSS thus uses exponentially decreasing batch sizes. The *Trapezoid Self-Scheduling (TSS)* strategy [7] also uses decreasing batch sizes, but the batch sizes decrease linearly from a first size f to a last size l. They advocate the use of $f = N/(2p)$ and $l = 1$. The *Factoring (Fac.)* and *Weighted Factoring (WF)* strategies also use decreasing batch sizes. At each time step, half of the remaining tasks are given out. The WF strategy works by assigning a weight to each processor corresponding to the computing speed of the processor before the computation starts, and allocates tasks based on these weights in every round [1,8].

3.2 The Monitor Strategy

The Monitor strategy is fairly similar to Weighted Factoring (WF), which assigns tasks to workers in a weighted fashion for each round, where each worker has a static weight. For WF, this weight has to be computed in advance, before the actual computations start, which is a disadvantage in a dynamic environment. The Monitor strategy, however, performs such benchmarking *online* throughout the computation and thus uses dynamic weights, which also allows for good performance in a truly dynamic environment. The strategy uses an initialization phase and several batch computation phases.

During the initialization phase, workers request tasks from the master which are handed out one by one. Workers measure the time it takes to compute their respective task, and report these timings to the master when they request another task. When all workers have reported their task computation times, the initialization phase is done.

Formally, let x_i be the number of tasks that worker w_i will be given in the current batch computation phase, and y_i be the number of uncomputed tasks queued by worker w_i. Let t_i denote the time taken to process one task, and T_i denote the time taken for worker w_i to finish the current phase. Recall that R denotes the number of unassigned tasks held by the master, and p denotes the number of workers. In a batch computation phase, the master starts by solving the following system of equations:

$$\begin{cases} (1) \;\; \forall i \in [0, p\rangle, \;\; T_i = (y_i + x_i) \times t_i \\ (2) \;\; \forall i \in [1, p\rangle, \;\; T_i = T_{i-1} \\ (3) \qquad\qquad\quad \sum_{i=0}^{p-1} x_i = R/2 \end{cases}$$

In a given phase, worker w_i receive x_i tasks that are added to the y_i uncomputed tasks already stored in its local task queue. It will hence finish its execution of the current phase at time $T_i = (y_i + x_i) \times t_i$ (equation 1). For the total execution time to be minimized, all workers must finish their computations simultaneously, hence $\forall i \in [1, p\rangle, T_i = T_{i-1}$ (equation 2). This condition has been proved in the context of Divisible Load Theory [9]. The sum of all x_i is equal to $R/2$, meaning that $R/2$ tasks are allocated during the current phase (equation 3). It has been found experimentally [8] that handing out half of the remaining tasks in each round gives good performance.

Throughout the computation, workers periodically (i.e. every r computed tasks) report their task computation times t_i and the number of uncomputed tasks y_i waiting in their local task queues to the master. Hence the master is continuously monitoring the worker states. Consequently, after the first batch computation phase, there is no need for another initialization phase, since the master has up-to-date knowledge of the performance of the workers. Note that the parameter r must be tuned for the application and cluster in use. As soon as a worker is done with its local tasks, a request is sent to the master, which then enters the next computation phase. A new system of equations is solved with the last up-to-date values of T_i and y_i.

Throughout the computation, y_i has a great significance. Suppose that at phase k, worker w_i has no external load, and can thus supply a large amount of computing power to our application. The master will then delegate a large number of tasks to w_i. Suppose now that during phase k, w_i receives a large external load, slowing down its task execution rate. At the end of phase k worker w_i will still have a lot of uncomputed tasks. The master has up-to-date knowledge of this, and allocates only a few (or no) new tasks to w_i in phase $k + 1$.

Note that if some workers are slowed down *drastically* the above system of equations may yield negative x_i values. Since the Monitor strategy does not consider withdrawing tasks from workers, the corresponding equations are removed, and the linear system is solved once more, distributing hence tasks among the remaining workers. This process is repeated until the solution yields no negative x_i values.

The task computation time t_i reported by worker w_i will typically be the mean value of its η last computation times. Having $\eta = 1$ might give a non-optimal allocation, since the timing can vary a lot in a dynamic environment. At the other end of the spectrum, a too high value for η conceals changes in processing speeds, which is also non-optimal. The parameter η needs to be adjusted for the individual application and/or environment.

Fig. 2 shows the allocated batch sizes for the scheduling strategies described in Section 3.1 as well as the Monitor strategy, when the processors report the task computation times shown in Fig. 3. Note that for the Monitor strategy, we assume $y_i = 0$ at the beginning of every phase, meaning that all the processors have computed all their assigned tasks from the previous phase.

Strategy	**Batch sizes**
SC	**128 128 128 128**
SS	**1 1 1 1** 1 1 1 1 **1 1 1 1** 1 1 1 1 **1 1 1 1** ...
GSS	**128 96 72 54** 40 30 23 17 **13 9 7 5** 4 3 2 2 **1 1 1 1** 1 1 1
TSS	**64 60 56 52** 48 44 40 36 **32 28 24 20** 8
Fac.	**64 64 64 64** 32 32 32 32 **16 16 16 16** 8 8 8 8 **4 4 4 4** 2 2 2 2 **1 1 1 1** 1 1 1 1
WF	**180 32 20 24** 90 16 10 12 **45 8 5 6** 22 4 3 3 **11 2 1 2** 6 1 0 1 **3 1 0 0** 1 1 0 0 **1 0 0 0** 1 0 0 0
Monitor	1 1 1 1 1 1 1 1 **177 32 20 23** 73 27 12 14 **11 23 13 16** 4 7 14 6 **6 3 3 4** 4 1 1 1 **0 2 1 1** 1 2 1 1

Fig. 2. Batch sizes for various sched. strat. with $N = 512$ tasks and $p = 4$ workers

4 Implementation

4.1 Data Staging

In order to improve I/O parallelism, the worker nodes read the necessary input data from the NFS disk themselves, much like the data staging technique presented in [10]. The

Processor	Task computation times at time steps								
	1	2	3	4	5	6	7	8	9
1	0.10	0.15	1.01	0.90	0.28	0.29	0.99	0.90	0.89
2	0.56	0.40	0.50	0.48	0.52	0.53	0.47	0.49	0.50
3	0.89	0.90	0.89	0.24	0.67	0.88	0.60	0.66	0.63
4	0.75	0.76	0.74	0.50	0.45	0.70	0.69	0.63	0.62

Fig. 3. Examples of task computation times for 4 processors at 9 time steps throughout a computation. Note that for WF, the times at step 1 are used as weights

master receives short requests from workers, and answer these requests with a short message containing only a pointer to the input data to be fetched, thus circumventing the master bottleneck.

However, because our test-case application has such a high computation to I/O ratio, our experiments showed that data staging did not have a great performance impact for this application [4]. Data staging is nevertheless a valuable technique for applications whose computation to I/O ratio is lower.

4.2 Multithreaded Processes

In order to avoid processor idleness, we decided to implement a multithreaded approach. Every worker process is composed of three threads: a main thread for communicating with the master, a thread for reading input data from disk, and a thread for computing output data. The main thread requests tasks and buffers them in a task queue until the number of tasks buffered is above a user defined threshold ϕ. It then goes to sleep and wakes up when the number of tasks in the queue is below ϕ. The goal of the main thread is to keep ϕ tasks queued at all times. The input reader thread will fetch tasks from the task queue, and, if the queue is empty, sleep while waiting for a task. Once the task queue is non-empty, the input reader thread will read and store input data, then add pointers to the data locations in the input data queue. And this, until the number of input data units in the queue is above ϕ. It then goes to sleep, and wakes up when the number of input data units in the queue is below ϕ. The procedure is then repeated. The computer thread works in much the same way as the input reader thread. See [4] for details.

The threshold ϕ regulates how soon the workers will request tasks from the master. Intuitively, $\phi = 1$ might be non-optimal, since the task computer thread might become idle while the main thread is waiting for the master to allocate a task. Note that since each worker has two queues of size ϕ, it buffers 2ϕ tasks.

The master process is also multi-threaded, with one thread continuously probing for, receiving and sending messages to/from workers using MPI, one thread executing the scheduling strategy in use, and one worker thread computing tasks on the master processor. The MPI thread terminates as soon as all tasks have been allocated, but until that point, it consumes quite a lot of CPU time that could have been used by the worker thread. This means that for the worker thread on the master, a high ϕ value is optimal, since the workers will request tasks quickly and the MPI thread will terminate early. This is a side effect of our non-optimal master architecture, since the MPI thread consumes unnecessary CPU power. One possible optimization would be to merge the thread for communicating through MPI and the thread for executing the scheduling strategy, but this would lead to non-modular code. Another possible optimization would be to use two threads calling MPI, one for receiving and one for sending, but this is impossible with the non-thread-safe MPICH library we had at our disposal. For more on this side effect, see [4].

5 Empirical Results and Analysis

The implemented scheduling strategies were compared for different values of ϕ on our dedicated cluster which is a static environment [4]. Our goal was to find the best scheduling strategy combined with the optimal parameters ϕ, η and r for the test-case application running on the test-bed cluster. Note that for the Monitor strategy, we experimentally found $r = 4$ and $\eta = 20$ to be optimal values for our application and cluster [4], and these values have been used in the following experiments. The results from our experiments are shown in Fig. 4; for more experiments, see [4].

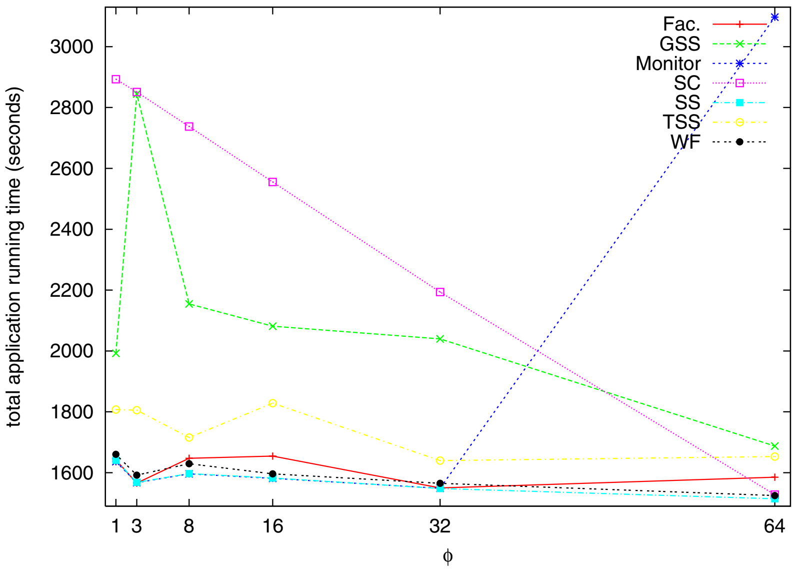


Fig. 4. Comparison of sched. strategies with increasing ϕ values, static environment

These experiments were conducted using 8 nodes, and an image of size 2048×2048 pixels decomposed into 1024 blocks, each block corresponding to a task. One interesting finding is that the Static Chunking strategy performs linearly better when using a larger ϕ value. When ϕ increases, the workers request new tasks earlier, hence causing the termination of the master MPI thread earlier. This frees resources for the worker thread on the master, and thus makes it process tasks faster. One might argue that the MPI thread on the master should have been terminated early regardless of ϕ since SC only uses one round of allocation, but in order to be fair, we kept the same master implementation for all the scheduling strategies. Consequently, all scheduling strategies must take into account the worker thread on the master which is very slow compared to the dedicated workers. Therefore, SC is a bad choice in a heterogeneous environment, while it is good in a homogeneous environment, as shown when $\phi = 64$.

The SS, Fac., WF and Monitor strategies are all quite similar. The reason why we get better results with $\phi = 32$ than with $\phi = 3$ is the same as for the SC case. The master MPI thread is stopped earlier, and we have one faster worker for the rest of the computation. With $\phi > 32$, the Monitor strategy performs very badly. One possible explanation for this is that during the initialization phase, the master computing thread is very slow, and will be given a small amount of tasks, less than ϕ. Consequently, the master computing thread will constantly request more tasks. As a result, the scheduling thread will solve a lot of unnecessary systems of equations further slowing down the computing thread.

Nevertheless, it should be noted that using very high ϕ values prevents good load balancing, since in order to allocate tasks when the workers need them (or slightly before, to avoid idle time), ϕ must be kept relatively low.

It is quite surprising that the Self-Scheduling strategy, which has the highest amount of communication of all strategies, is among the very fastest scheduling strategies. A possible explanation is that our multi-threaded implementation is able to hide the communication delays, and because our application has a high computation to I/O ratio. However, our environment is relatively homogeneous and static, and we expect the Monitor strategy to outperform SS in a strongly heterogeneous and dynamic environment.

Fig. 5 shows speedup results. Note that using e.g. 2 processors means using the master with its separate worker thread and 1 dedicated worker. With a 512×512-pixel image, the speedup drops significantly when using more than 4 processors. This is due to using a suboptimal task size for this relatively small image size [4]. For the larger image sizes, the speedup increases when adding more processors. Intuitively, this comes from the fact that the relatively slow worker thread of the master processor plays a smaller role when adding more processors. With a sufficiently large image and 8 or more processors, we have a close to linear speedup.

6 Conclusions and Future Work

A novel online scheduling strategy has been designed and implemented. The Monitor strategy was experimentally compared to implementations of six other popular scheduling strategies found in the literature. Experiments show that the Monitor strategy performs excellently compared to these strategies, and should be especially well suited for dynamic environments.

The monitor strategy implementation involved multi-threading and data staging, two techniques that decrease processor idle time and increase master utilization. Our test-case application, the matched filtering algorithm, has a very high computation to I/O ratio, and consequently data staging is probably unnecessary for this application. Experimental tests on our cluster show, however, that data staging is a valuable technique for

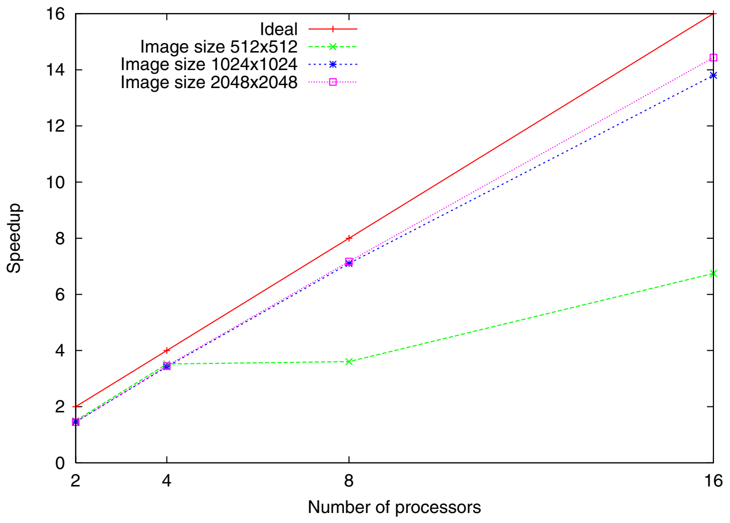


Fig. 5. The speedup of the application

applications whose computation to I/O ratio is lower. The multi-threaded implementation of the worker processes, using task queuing mechanisms, is able to hide communication delays and keep the workers processing data continuously. The excellent performance of both the Self-Scheduling and the Monitor strategy substantiate this.

This work could be extended in the following directions. First, running more experiments on other clusters which provide more heterogeneity and more dynamism would enable to measure the potential of the Monitor strategy.

Second, the Monitor strategy has three user-specifiable parameters, r, η and ϕ. A way to determine the optimal values of these parameters online would be desirable. It might be that an optimal solution in heterogeneous environment necessitate different values of these parameters for different workers.

Then, the optimal task size used in this study has been found experimentally, but it would be desirable to determine it online: Two procedures for doing this are discussed in [4].

Finally, a thread-safe MPI library would enable us to implement the master process differently [4], which would increase the performance.

References

1. Hummel, S.F., Schonberg, E., Flynn, L.E.: Factoring: A method for scheduling parallel loops. Comm. of the ACM **35** (1992) 90–101
2. Basney, J., Raman, R., Livny, M.: High Throughput Monte Carlo. In: Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing. (1999)
3. Chaudhuri, S., Chatterjee, S., Katz, N., Nelson, M., Goldbaum, M.: Detection of blood vessels in retinal images using two-dimensional matched filters. IEEE Trans. on Med. Imaging **8** (1989) 263–269

4. Rosenvinge, E.M.R.: Online Task Scheduling On Heterogeneous Clusters: An Experimental Study. Master's thesis, NTNU (2004) `http://www.idi.ntnu.no/~elster/students/ms-theses/rosenvinge-msthesis.pdf`.
5. Kruskal, C.P., Weiss, A.: Allocating independent subtasks on parallel processors. IEEE Trans. on Software Eng. **11** (1985) 1001–1016
6. Polychronopoulos, C.D., Kuck, D.J.: Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. IEEE Trans. on Comp. **36** (1987) 1425–1439
7. Tzen, T.H., Ni, L.M.: Dynamic loop scheduling for shared-memory multiprocessors. In: Proc. of the 1991 Int'l Conference on Parallel Processing, IEEE Computer Society (1991) II247–II250
8. Hummel, S.F., Schmidt, J., Uma, R.N., Wein, J.: Load-sharing in heterogeneous systems via weighted factoring. In: Proceedings of the eighth annual ACM symposium on Parallel algorithms and architectures, ACM Press (1996) 318–328
9. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.G.: Scheduling Divisible Loads in Parallel and Distributed Systems. Computer Society (1996)
10. Elwasif, W., Plank, J.S., Wolski, R.: Data staging effects in wide area task farming applications. In: IEEE Int'l Symposium on Cluster Computing and the Grid, Brisbane, Australia (2001) 122–129

A Parallel Method for Large Sparse Generalized Eigenvalue Problems by OmniRPC in a Grid Environment

Tetsuya Sakurai[1,3], Kentaro Hayakawa[2], Mitsuhisa Sato[1], and Daisuke Takahashi[1]

[1] Department of Computer Science
University of Tsukuba, Tsukuba 305-8573, Japan
{sakurai,msato,daisuke}@cs.tsukuba.ac.jp
[2] Doctoral Program of Systems and Information Engineering
University of Tsukuba, Tsukuba 305-8573, Japan
hayakawa@nalab.cs.tsukuba.ac.jp
[3] Core Research for Evolutional Science and Technology (CREST)

Abstract. In this paper we present a parallel method for finding several eigenvalues and eigenvectors of a generalized eigenvalue problem $A\boldsymbol{x} = \lambda B\boldsymbol{x}$, where A and B are large sparse matrices. A moment-based method by which to find all of the eigenvalues that lie inside a given domain is used. In this method, a small matrix pencil that has only the desired eigenvalues is derived by solving large sparse systems of linear equations constructed from A and B. Since these equations can be solved independently, we solve them on remote hosts in parallel. This approach is suitable for master-worker programming models. We have implemented and tested the proposed method in a grid environment using a grid RPC (remote procedure call) system called OmniRPC. The performance of the method on PC clusters that were used over a wide-area network was evaluated.

1 Introduction

The generalized eigenvalue problem

$$A\boldsymbol{x} = \lambda B\boldsymbol{x},$$

where A and B are $n \times n$ real or complex matrices, arises in many scientific applications. In such applications the matrix is often large and sparse, and we may need to find a number of the eigenvalues that have some special property and their corresponding invariant subspaces.

Several methods for such eigenvalue problems are building sequences of subspaces that contain the desired eigenvectors. Krylov subspace based techniques are powerful tools for large-scale eigenvalue problems [1,2,9,10]. The relations among Krylov subspace methods, moment-matching approach and Padé approximation are shown in [2].

In this paper, we present a parallel method for finding several eigenvalues and eigenvectors of a generalized eigenvalue problem. A moment-based method to find all of the eigenvalues that lie inside a given domain is used. In this method, a small matrix pencil that has only the desired eigenvalues is derived by solving large sparse linear equations constructed from A and B. Because these equations can be solved independently, we solve them on remote hosts in parallel. In this approach, we do not need to exchange data between remote hosts. Therefore, the presented method is suitable for master-worker programming models. Moreover, the method has a good load balancing property.

We have implemented and tested our method in a grid environment using a grid RPC (remote procedure call) system called OmniRPC [8,13]. OmniRPC is a grid RPC system that allows seamless parallel programming in both cluster and grid environments and supports remote hosts with "ssh", as well as a grid environment with Globus.

The performance of the presented method on PC clusters used over a wide-area network was evaluated. As a test problem, we used the matrices that arise in the calculation of the electronic state of molecular hydrogen. The results show that the method is efficient in a grid environment.

2 A Moment-Based Method

Let $A, B \in \mathbb{C}^{n\times n}$, and let $\lambda_1, \dots, \lambda_d$ ($d \le n$) be finite eigenvalues of the matrix pencil $A - \lambda B$. The pencil $A - \lambda B$ is referred to as regular if $\det(A - \lambda B)$ is not identically zero for $\lambda \in \mathbb{C}$.

For nonzero vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{C}^n$, we define

$$f(z) := \boldsymbol{u}^H (zB - A)^{-1} \boldsymbol{v}.$$

The function $f(z)$ is analytic when $zB - A$ is nonsingular.

Suppose that m distinct eigenvalues $\lambda_1, \dots, \lambda_m$ are located inside a positively oriented closed Jordan curve Γ. We consider the problem of finding all of the poles of $f(z)$ inside Γ. Define the complex moments

$$\mu_k := \frac{1}{2\pi \mathrm{i}} \int_\Gamma (z - \gamma)^k f(z) dz, \quad k = 0, 1, \dots, \tag{2.1}$$

where γ is located inside Γ. Let the $m \times m$ Hankel matrices H_m and $H_m^<$ be $H_m := [\mu_{i+j-2}]_{i,j=1}^m$ and $H_m^< := [\mu_{i+j-1}]_{i,j=1}^m$. Then we have the following theorem ([12]).

Theorem 1. *Let the pencil $A - \lambda B$ be regular. Then the eigenvalues of the pencil $H_m^< - \lambda H_m$ are given by $\lambda_1 - \gamma, \dots, \lambda_m - \gamma$.*

Therefore, we can obtain the eigenvalues $\lambda_1, \dots, \lambda_m$ by solving the generalized eigenvalue problem $H_m^< \boldsymbol{x}' = \lambda H_m \boldsymbol{x}'$, where $\boldsymbol{x}' \in \mathbb{C}^m$. If we find an appropriate Γ that includes a small number of eigenvalues, then the derived eigenproblem is small.

This approach is based on the root finding method for an analytic function proposed in [6]. This method finds all of the zeros that lie in a circle using numerical integration. Error analyses for the eigenvalues of the pencil $H_m^< - \lambda H_m$ of which elements are obtained by numerical integration are presented in [7] and [11].

Let $\boldsymbol{s}_k$ be

$$\boldsymbol{s}_k := \frac{1}{2\pi \mathrm{i}} \int_\Gamma (z - \gamma)^k (zB - A)^{-1} \boldsymbol{v} dz, \quad k = 0, 1, \dots, \tag{2.2}$$

and let $\boldsymbol{q}_1, \dots, \boldsymbol{q}_m$ be eigenvectors corresponding to the eigenvalues $\lambda_1, \dots, \lambda_m$. From the results of [7] and [12], we have the following result.

Theorem 2. *Let W_m be an $m \times m$ matrix such that*

$$H_m^< W_m = \mathrm{diag}(\zeta_1, \ldots, \zeta_m) H_m W_m,$$

where $\zeta_j = \lambda_j - \gamma$, $j = 1, \ldots, m$. Then

$$[\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m] = [\boldsymbol{s}_0, \ldots, \boldsymbol{s}_{m-1}] W_m. \tag{2.3}$$

Let $\nu_1, \ldots, \nu_m$ be residues of $\zeta_1, \ldots, \zeta_m$. Then

$$\mu_k = \sum_{j=1}^{m} \nu_j \zeta_j^k, \quad k = 0, 1, \ldots. \tag{2.4}$$

It follows from (2.4) that

$$(\mu_0, \ldots, \mu_{m-1}) = (\nu_1, \ldots, \nu_m) V_m,$$

where V_m is the $m \times m$ Vandermonde matrix $V_m := [\zeta_i^{j-1}]_{i,j=1}^m$.

Since the column vectors of V_m^{-1} are given by the eigenvectors of the matrix pencil $H_m^< - \lambda H_m$ ([7]), the residues ν_j can be evaluated by

$$(\nu_1, \ldots, \nu_m) = (\mu_0, \ldots, \mu_{m-1}) V_m^{-1} = (\mu_0, \ldots, \mu_{m-1}) W_m. \tag{2.5}$$

We next consider the case in which Γ is given by a circle and the integration is evaluated via a trapezoidal rule on the circle. Let γ and ρ be the center and the radius, respectively, of the given circle. Let N be a positive integer, and let

$$\omega_j := \gamma + \rho e^{\frac{2\pi \mathrm{i}}{N}(j+1/2)}, \quad j = 0, 1, \ldots, N-1.$$

By approximating the integral of equation (2.1) via the N-point trapezoidal rule, we obtain the following approximations for μ_k:

$$\mu_k \approx \hat{\mu}_k := \frac{1}{N} \sum_{j=0}^{N-1} (\omega_j - \gamma)^{k+1} f(\omega_j), \quad k = 0, 1, \ldots \tag{2.6}$$

Let the $m \times m$ Hankel matrices $\hat{H}_m$ and $\hat{H}_m^<$ be $\hat{H}_m := [\hat{\mu}_{i+j-2}]_{i,j=1}^m$ and $\hat{H}_m^< := [\hat{\mu}_{i+j-1}]_{i,j=1}^m$. Let $\hat{\zeta}_1, \ldots, \hat{\zeta}_m$ be the eigenvalues of the matrix pencil $\hat{H}_m^< - \lambda \hat{H}_m$. We regard $\hat{\lambda}_j = \gamma + \hat{\zeta}_j$, $1 \le j \le m$ as the approximations for $\lambda_1, \ldots, \lambda_m$.

Let $\hat{W}_m$ be the matrix of which column vectors are eigenvectors of $\hat{H}_m^< - \lambda \hat{H}_m$. The approximate vectors $\hat{\boldsymbol{q}}_1, \ldots, \hat{\boldsymbol{q}}_m$ for the eigenvectors $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_m$ are obtained by

$$[\hat{\boldsymbol{q}}_1, \ldots, \hat{\boldsymbol{q}}_m] = [\hat{\boldsymbol{s}}_0, \ldots, \hat{\boldsymbol{s}}_{m-1}] \hat{W}_m, \tag{2.7}$$

where

$$\boldsymbol{y}_j = (\omega_j B - A)^{-1} \boldsymbol{v}, \quad j = 0, 1, \ldots, N-1,$$

and

$$\hat{\boldsymbol{s}}_k := \frac{1}{N} \sum_{j=0}^{N-1} (\omega_j - \gamma)^{k+1} \boldsymbol{y}_j, \quad k = 0, 1, \ldots. \tag{2.8}$$

We then obtain the following algorithm:

Algorithm:

```
Input: $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{C}^n$, $N$, $m$, $\gamma$, $\rho$
Output: $\hat{\lambda}_1, \ldots, \hat{\lambda}_m$, $\hat{\boldsymbol{q}}_1, \ldots, \hat{\boldsymbol{q}}_m$
1. Set $\omega_j \leftarrow \gamma + \rho \exp(2\pi \mathrm{i}(j+1/2)/N), j = 0, \ldots, N-1$
2. Solve $(\omega_j B - A)\boldsymbol{y}_j = \boldsymbol{v}$ for $\boldsymbol{y}_j, j = 0, \ldots, N-1$
3. Set $f(\omega_j) \leftarrow \boldsymbol{u}^H \boldsymbol{y}_j, j = 0, \ldots, N-1$
4. Compute $\hat{\mu}_k, k = 0, \ldots, 2m-1$ by (2.6)
5. Compute the eigenvalues $\hat{\zeta}_1, \ldots, \hat{\zeta}_m$ of the pencil $\hat{H}_m^< - \lambda \hat{H}_m$
6. Compute $\hat{\boldsymbol{q}}_1, \ldots, \hat{\boldsymbol{q}}_m$ by (2.7)
7. Set $\hat{\lambda}_j \leftarrow \gamma + \hat{\zeta}_j, j = 1, \ldots, m$
```

In practical use, the exact number of eigenvalues in the circle is not given in advance. The criteria to find appropriate m were discussed in [3,6] for the Hankel matrices. If we take m larger than the exact number of eigenvalues, then the set of eigenvalues of the matrix pencil $\hat{H}_m^< - \lambda \hat{H}_m$ includes approximations of the eigenvalues in Γ.

3 Parallel Implementation on OmniRPC

In this section we describe a parallel implementation of the algorithm. To evaluate the value of $f(z)$ at $z = \omega_j, j = 0, \ldots, N-1$, we solve the systems of linear equations

$$(\omega_j B - A)\boldsymbol{y}_j = \boldsymbol{v}, \quad j = 0, 1, \ldots, N-1. \tag{3.9}$$

When the matrices A and B are large and sparse, the computational costs to solve the linear systems (3.9) are dominant in the algorithm. Since these linear systems are independent, we solve them on remote hosts in parallel.

We have implemented the algorithm in a grid RPC system called OmniRPC. OmniRPC efficiently supports typical master-worker parallel grid applications such as parametric execution programs. The user can define an initialization procedure in the remote executable to send and store data automatically in advance of actual remote procedure calls in `OmniRPC_Module_Init`.

`OmniRPC_Call` is a simple client programming interface for calling remote functions. When `OmniRPC_Call` makes a remote procedure call, the call is allocated to an appropriate remote host. The programmer can use asynchronous remote procedure calls by

$$OmniRPC_Call_Async.$$

The procedure to solve N systems in Step 2 of the presented algorithm is performed by `OmniRPC_Call_Async` in the source code. Since A, B and $\boldsymbol{v}$ are common in each system of linear equations, we only need to send these data at the first time to each host. To solve another equation on a remote host, a scalar parameter ω_j is sent. If we do not need eigenvectors, the scalar value $f(\omega_j)$ is returned from a remote host after the procedure in Step 3 is performed. Otherwise, $\boldsymbol{y}_j$ is returned in addition.

We can easily extend the method for the case that several circular regions are given. Suppose that N_c circles are given. Then we solve $N \times N_c$ systems of linear equations

$$(\omega_j^{(k)} B - A)\boldsymbol{y}_j^{(k)} = \boldsymbol{v}, \quad j = 0, \ldots, N-1, k = 1, \ldots, N_c,$$

where $\omega_j^{(k)}$, $j = 0, \ldots, N-1$ are points on the kth circle.

Now we consider the efficiency loss caused by the communication of the matrix data to the remote host. Let α be the ratio of the time to send matrix data to a remote host and the total time to solve all systems of linear equations. We assume that matrix data are sent to each remote host one after another, and the computation starts after receiving matrix data. We also assume that load imbalance does not occur in the computation. Then from a simple observation, we have the following estimation of the speedup:

$$S_p \approx n_p \Big/ \Big(1 - \alpha + \frac{\alpha}{2} n_p (n_p + 1)\Big),$$

where n_p is the number of remote hosts.

4 Numerical Examples

In this section, we present numerical examples of the proposed method. The algorithm was implemented in OmniRPC on a Linux operating system. Computation was performed with double precision arithmetic in FORTRAN.

The remote hosts were Intel Xeon computers with a clock speed of 2.4 GHz, and the local host was an Intel Celeron computer with a clock speed of 2.4 GHz and 100Base-T Ethernet. The local host and the remote hosts were connected over a wide-area network.

To evaluate the speed of execution for the parallel algorithm, the elapsed wall-clock time (in seconds) was observed. The time for loading input matrices into a memory of the local host and the time required to start up all OmniRPC processes were not included.

When A and B are real matrices, the relation $f(\overline{z}) = \overline{f(z)}$ holds. Thus we evaluated $N/2$ function values $f(\omega_0), \ldots, f(\omega_{N/2-1})$, and set the remaining function values using

$$f(\omega_{N-1-j}) = \overline{f(\omega_j)}.$$

The elements of $\boldsymbol{u}$ and $\boldsymbol{v}$ were distributed randomly on the interval $[0, 1]$ by a random number generator, and they were calculated on the remote hosts.

Example 1. The matrices were

$$A = I_n, \quad B = \begin{pmatrix} 5 & -4 & 1 & & & & \\ -4 & 6 & -4 & 1 & & & \\ 1 & -4 & 6 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & -4 & 6 & -4 & 1 \\ & & & 1 & -4 & 6 & -4 \\ & & & & 1 & -4 & 5 \end{pmatrix},$$

where I_n is the $n \times n$ identity matrix, and $n = 2000000$.

The exact eigenvalues are given by

$$\lambda_j^* = \frac{1}{16 \cos^4\left(\frac{j\pi}{2(n+1)}\right)}, \quad j = 1, 2, \ldots, n.$$

The linear systems were solved by a direct solver for a band matrix. The interval $[2, 2.005]$ was covered by 100 circles with radius $\rho = 2.5 \times 10^{-5}$. The number of points on the circle was $N = 32$. The size of the Hankel matrices was $m = 10$, and we selected eigenvalues in each circle if the corresponding residue satisfies $|\nu_j| \geq 10^{-8}$. 368 eigenvalues in the interval were obtained.

The resulting eigenvalues and their errors were as follows:

j	λ_j	$\lvert\lambda_j - \lambda_j^*\rvert$
1	2.000012164733324	6.22E-15
2	2.000025723848086	6.22E-15
3	2.000039283082696	1.15E-14
$\vdots$	$\vdots$	$\vdots$
368	2.004996418973247	8.88E-16

In Table 1, the wall-clock times in seconds, the speedup, and S_p are shown. In this example, we set $\alpha = 0.0052$ from the results of the test program. The speedup was 7.66 with 10 remote processors, and we can see that S_p is a good estimation of the speedup.

Table 1. Wall-clock times in seconds, speedup and S_p in Example 1

Number of nodes	Time (sec)	Speedup	S_p
1	1149	1.00	1.00
2	583	1.97	1.98
4	304	3.78	3.82
6	214	5.37	5.43
8	174	6.60	6.76
10	150	7.66	7.80

Example 2. The test matrices were derived from the finite element method for a molecular electronic state described in [4]. Both A and B were real symmetric, and $n = 12173$. The number of nonzero elements was 509901.

We computed 15 eigenvalues and corresponding eigenvectors in 8 circles. The circles were located at $\gamma = -6.2, -5.8, -5.6, -3.4, -3.0, -2.8, -2.6, -2.0$ with radius $\rho = 0.1$. The parameters were chosen as $N = 32$ and $m = 10$.

Since the matrix $\omega_j B - A$ with complex ω_j is complex symmetric, the COCG method [14] with incomplete Cholesky decomposition was used to solve the linear equations. The stopping criterion for the relative residual was 10^{-12}.

In Table 2, the wall-clock times in seconds, the speedup, and S_p with $\alpha = 0.0045$ are shown. The speedup was similar to Example 1. While the computational time to solve the systems of linear equations were different for each ω_j, S_p was still a good

Table 2. Wall-clock times in seconds, speedup and S_p in Example 2

Number of nodes	Time (sec)	Speedup	S_p
1	199	1.00	1.00
2	103	1.93	1.98
4	53	3.75	3.84
6	38	5.24	5.50
8	29	6.86	6.91
10	26	7.65	8.04

estimation of the speedup. The reason that the speedup was slightly smaller than S_p was load imbalance.

If we give more circles or linear systems require more computational time then α becomes smaller. In such case, the speedup will increase.

5 Conclusions

In this paper we presented a parallel algorithm to find eigenvalues and eigenvectors of generalized eigenvalue problems using a moment-based method. In this approach, we do not need to exchange data between remote hosts. Therefore, the presented method is suitable for master-worker programming models.

We have implemented and tested the proposed method in a grid environment using a grid RPC system called OmniRPC. The remote procedures have large granularity, and the method has a good load balancing property.

The computation with explicit moments is often numerically unstable. Thus it is important to find appropriate parameters about circular regions. In some cases, a rough estimation of distribution of eigenvalues is given in advance by physical properties or some approximate methods. For example, eigenvalues appeared in computation of molecular orbitals are approximated by fragmented molecular orbitals [5]. The estimation of suitable parameters will be the subject of our future study.

Acknowledgements

This work was partially supported by CREST, Japan Science and Technology Agency (JST), and Grant-in-Aid for Scientific Research No. 15560049 from the Ministry of Education, Culture, Sports Science and Technology of Japan.

References

1. W. E. Arnoldi, The principle of minimized iteration in the solution of the matrix eigenproblem, *Quarterly of Appl. Math.*, 9:17–29, 1951.
2. Z. Bai, Krylov subspace techniques for reduced-order modeling of large-scale dynamical systems, *Appl. Numer. Math.*, 43:9–44, 2002.
3. G. H. Golub, P. Milanfar and J. Varah, A stable numerical method for inverting shape from moments, *SIAM J. Sci. Comp.*, 21(4):1222–1243, 1999.
4. S. Hyodo, Meso-scale fusion: A method for molecular electronic state calculation in inhomogeneous materials, Proc. the 15th Toyota Conference, Mikkabi, 2001, in *Special issue of J. Comput. Appl. Math.*, 149:101–118, 2002.
5. Yuichi Inadomi, Tatsuya Nakano, Kazuo Kitaura and Umpei Nagashima, Definition of molecular orbitals in fragment molecular orbital method, *Chemical Physics Letters*, 364:139–143, 2002.
6. P. Kravanja, T. Sakurai and M. Van Barel, On locating clusters of zeros of analytic functions, *BIT*, 39:646–682, 1999.
7. P. Kravanja, T. Sakurai, H. Sugiura and M. Van Barel, A perturbation result for generalized eigenvalue problems and its application to error estimation in a quadrature method for computing zeros of analytic functions, *J. Comput. Appl. Math.*, 161:339–347, 2003.
8. OmniRPC: http://www.omni.hpcc.jp/OmniRPC.
9. A. Ruhe, Rational Krylov algorithms for nonsymmetric eigenvalue problems II: matrix pairs, *Linear Algevr. Appl.*, 197:283–295, 1984.
10. Y. Saad, *Iterative Methods for Large Eigenvalue Problems*, Manchester University Press, Manchester, 1992.
11. T. Sakurai, P. Kravanja, H. Sugiura and M. Van Barel, An error analysis of two related quadrature methods for computing zeros of analytic functions, *J. Comput. Appl. Math.*, 152:467–480, 2003.
12. T. Sakurai and H. Sugiura, A projection method for generalized eigenvalue problems, *J. Comput. Appl. Math.*, 159:119–128, 2003.
13. M. Sato, T. Boku and D. Takahashi, OmniRPC: a Grid RPC System for parallel programming in cluster and grid environment, *Proc. CCGrid 2003*, 206–213, 2003.
14. H. A. van der Vorst, J. B. M. Melissen, A Petrov-Galerkin type method for solving $Ax = b$, where A is a symmetric complex matrix, *IEEE Trans. on Magnetics*, 26(2):706–708, 1990.

An Implementation of Parallel 3-D FFT Using Short Vector SIMD Instructions on Clusters of PCs

Daisuke Takahashi, Taisuke Boku, and Mitsuhisa Sato

Graduate School of Systems and Information Engineering, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki 305-8573, Japan
{daisuke,taisuke,msato}@cs.tsukuba.ac.jp

Abstract. In this paper, we propose an implementation of a parallel three-dimensional fast Fourier transform (FFT) using short vector SIMD instructions on clusters of PCs. We vectorized FFT kernels using Intel's Streaming SIMD Extensions 2 (SSE2) instructions. We show that a combination of the vectorization and block three-dimensional FFT algorithm improves performance effectively. Performance results of three-dimensional FFTs on a dual Xeon 2.8 GHz PC SMP cluster are reported. We successfully achieved performance of over 5 GFLOPS on a 16-node dual Xeon 2.8 GHz PC SMP cluster.

1 Introduction

The fast Fourier transform (FFT) [1] is an algorithm widely used today in science and engineering. Parallel three-dimensional FFT algorithms on distributed-memory parallel computers have been well studied [2,3,4,5,6].

Today, many processors have short vector SIMD instructions, e.g., Intel's SSE/SSE2/SSE3, AMD's 3DNow!, and Motorola's AltiVec. These instructions provide substantial speedup for digital signal processing applications. Efficient FFT implementations with short vector SIMD instructions have also been well studied [7,8,9,10,11,12].

Many FFT algorithms work well when the data sets fit into a cache. When a problem size exceeds the cache size, however, the performance of these FFT algorithms decreases dramatically. The key issue of the design for large FFTs is to minimize the number of cache misses.

Thus, both vectorization and high cache utilization are particularly important for achieving high performance on processors that have short vector SIMD instructions.

In this paper, we propose an implementation of a parallel three-dimensional FFT using short vector SIMD instructions on clusters of PCs.

Our proposed parallel three-dimensional FFT algorithm is based on the multicolumn FFT algorithm [13,14]. Conventional three-dimensional FFT algorithms require three multicolumn FFTs and three data transpositions. The three transpose steps typically are the chief bottlenecks in cache-based processors.

Some previous three-dimensional FFT algorithms [14,15] separate the multicolumn FFTs from the transpositions.

Taking the opposite approach, we combine the multicolumn FFTs and transpositions to reduce the number of cache misses, and we modify the conventional three-dimensional

FFT algorithm to reuse data in the cache memory. We use the block three-dimensional FFT algorithm to implement the parallel three-dimensional FFT algorithm.

We have implemented the parallel block three-dimensional FFT algorithm on a 16-node dual Xeon PC SMP cluster, and in this paper we report the performance results.

Section 2 describes vectorization of FFT kernels. Section 3 describes a block three-dimensional FFT algorithm used for problems that exceed the cache size. Section 4 gives a parallel block three-dimensional FFT algorithm. Section 5 describes the in-cache FFT algorithm used for problems that fit into a data cache. Section 6 gives performance results. In section 7, we provide some concluding remarks.

2 Vectorization of FFT Kernels

Streaming SIMD Extensions 2 (SSE2) were introduced into the IA-32 architecture in the Pentium 4 and Intel Xeon processors [16]. These extensions were designed to enhance the performance of IA-32 processors.

The most direct way to use the SSE2 instructions is to inline the assembly language instructions into source code. However, this can be time-consuming and tedious, and assembly language inline programming is not supported on all compilers. Instead, Intel provides easy implementation through the use of API extension sets referred to as intrinsics [17].

We used the SSE2 intrinsics to access SIMD hardware. An example of complex multiplication using the SSE2 intrinsics is shown in Fig. 1.

The `__m128d` data type in Fig. 1 is supported by the SSE2 intrinsics. The `__m128d` data type holds two packed double-precision floating-point values. In the complex multiplication, the `__m128d` data type is used as a double-precision complex data type.

The inline function `ZMUL` in Fig. 1 can be used to multiply two double-precision complex values. To add two double-precision complex values, we can use the intrinsic function `__mm_add_pd` in Fig. 1.

To vectorize FFT kernels, the SSE2 intrinsics and the inline function `ZMUL` can be used. An example of a vectorized radix-2 FFT kernel using the SSE2 intrinsics is shown in Fig. 2.

3 A Block Three-Dimensional FFT Algorithm

The three-dimensional discrete Fourier transform (DFT) is given by

$$y(k_1, k_2, k_3) = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \sum_{j_3=0}^{n_3-1} x(j_1, j_2, j_3) \omega_{n_3}^{j_3 k_3} \omega_{n_2}^{j_2 k_2} \omega_{n_1}^{j_1 k_1}, \qquad (3.1)$$

where $\omega_{n_r} = e^{-2\pi i/n_r}$ $(1 \leq r \leq 3)$ and $i = \sqrt{-1}$.

The three-dimensional FFT based on the multicolumn FFT algorithm is as follows:

Step 1: Transpose

$$x_1(j_3, j_1, j_2) = x(j_1, j_2, j_3).$$

```
#include <emmintrin.h>

__m128d ZMUL(__m128d a, __m128d b);

static __inline __m128d ZMUL(__m128d a, __m128d b)
{
  __m128d ar, ai;

  ar = _mm_unpacklo_pd(a, a);           /* ar = [a.r a.r] */
  ar = _mm_mul_pd(ar, b);               /* ar = [a.r*b.r a.r*b.i] */
  ai = _mm_unpackhi_pd(a, a);           /* ai = [a.i a.i] */
  ai = _mm_xor_pd(ai, _mm_set_sd(-0.0)); /* ai = [-a.i a.i] */
  b = _mm_shuffle_pd(b, b, 1);          /* b = [b.i b.r] */
  ai = _mm_mul_pd(ai, b);               /* ai = [-a.i*b.i a.i*b.r] */

  return _mm_add_pd(ar, ai);            /* [a.r*b.r-a.i*b.i a.r*b.i+a.i*b.r] */
}
```

Fig. 1. An example of double-precision complex multiplication using SSE2 intrinsics

Step 2: n_1n_2 individual n_3-point multicolumn FFTs

$$x_2(k_3, j_1, j_2) = \sum_{j_3=0}^{n_3-1} x_1(j_3, j_1, j_2)\omega_{n_3}^{j_3k_3}.$$

Step 3: Transpose

$$x_3(j_2, j_1, k_3) = x_2(k_3, j_1, j_2).$$

Step 4: n_1n_3 individual n_2-point multicolumn FFTs

$$x_4(k_2, j_1, k_3) = \sum_{j_2=0}^{n_2-1} x_3(j_2, j_1, k_3)\omega_{n_2}^{j_2k_2}.$$

Step 5: Transpose

$$x_5(j_1, k_2, k_3) = x_4(k_2, j_1, k_3).$$

Step 6: n_2n_3 individual n_1-point multicolumn FFTs

$$y(k_1, k_2, k_3) = \sum_{j_1=0}^{n_1-1} x_5(j_1, k_2, k_3)\omega_{n_1}^{j_1k_1}.$$

The distinctive features of the three-dimensional FFT can be summarized as:

– Three multicolumn FFTs are performed in steps 2, 4 and 6. Each column FFT is small enough to fit into the data cache.
– The three-dimensional FFT has the three transpose steps, which typically are the chief bottlenecks in cache-based processors.

We combine the multicolumn FFTs and transpositions in order to reduce the number of cache misses, and we modify the conventional three-dimensional FFT algorithm to reuse data in the cache memory. As in the conventional three-dimensional FFT above, it is assumed in the following that $n = n_1n_2n_3$ and that n_b is the block size. We assume that each processor has a multilevel cache memory. A block three-dimensional FFT algorithm [6] can be stated as follows.

```
#include <emmintrin.h>

__m128d ZMUL(__m128d a, __m128d b);

void fft_vec(double *a, double *b, double *w, int m, int l)
{
  int i, i0, i1, i2, i3, j;
  __m128d t0, t1, w0;

  for (j = 0; j < l; j++) {
    w0 = _mm_load_pd(&w[j << 1]);
    for (i = 0; i < m; i++) {
      i0 = (i << 1) + (j * m << 1); i1 = i0 + (m * l << 1);
      i2 = (i << 1) + (j * m << 2); i3 = i2 + (m << 1);
      t0 = _mm_load_pd(&a[i0]); t1 = _mm_load_pd(&a[i1]);
      _mm_store_pd(&b[i2], _mm_add_pd(t0, t1));
      _mm_store_pd(&b[i3], ZMUL(w0, _mm_sub_pd(t0, t1)));
    }
  }
}
```

Fig. 2. An example of a vectorized radix-2 FFT kernel using SSE2 intrinsics

1. Consider the data in main memory as an $n_1 \times n_2 \times n_3$ complex matrix. Fetch and transpose the data n_b rows at a time into an $n_3 \times n_b$ matrix. The $n_3 \times n_b$ array fits into the L2 cache.
2. For each n_b column, perform n_b individual n_3-point multicolumn FFTs on the $n_3 \times n_b$ array in the L2 cache. Each column FFT also fits into the L1 data cache.
3. Transpose each of the resulting $n_3 \times n_b$ matrices, and return the resulting n_b rows to the same locations in the main memory from which they were fetched.
4. Fetch and transpose the data n_b rows at a time into an $n_2 \times n_b$ matrix.
5. For each n_b column, perform n_b individual n_2-point multicolumn FFTs on the $n_2 \times n_b$ array in the L2 cache.
6. Transpose each of the resulting $n_2 \times n_b$ matrices, and return the resulting n_b rows to the same locations in the main memory from which they were fetched.
7. Perform $n_2 n_3$ individual n_1-point multicolumn FFTs on the $n_1 \times n_2 \times n_3$ array.

We note that this algorithm is a *three-pass* algorithm. Fig. 3 gives the Fortran program for this block three-dimensional FFT algorithm. Here, the arrays `YWORK` and `ZWORK` are the work array. The parameters `NB` and `NP` are the blocking parameter and padding parameter, respectively.

4 Parallel Block Three-Dimensional FFT Algorithm

Let $N = N_1 \times N_2 \times N_3$. On a distributed-memory parallel computer which has P processors, the three-dimensional array $x(N_1,\, N_2,\, N_3)$ is distributed along the first dimension, N_1. If N_1 is divisible by P, each processor has distributed data of size N/P. We introduce the notation $\hat{N}_r \equiv N_r/P$ and we denote the corresponding index as $\hat{J}_r$, which indicates that the data along J_r are distributed across all P processors. Here, we use the subscript r to indicate that this index belongs to dimension r. The distributed array is represented as $\hat{x}(\hat{N}_1,\, N_2,\, N_3)$. At processor m, the local index $\hat{J}_r(m)$ corresponds

```
COMPLEX*16 X(N1,N2,N3)
COMPLEX*16 YWORK(N2+NP,NB),ZWORK(N3+NP,NB)
DO J=1,N2
  DO II=1,N1,NB
    DO KK=1,N3,NB
      DO I=II,MIN0(II+NB-1,N1)
        DO K=KK,MIN0(KK+NB-1,N3)
          ZWORK(K,I-II+1)=X(I,J,K)
        END DO
      END DO
    END DO
    DO I=II,MIN0(II+NB-1,N1)
      CALL IN_CACHE_FFT(ZWORK(1,I-II+1),N3)
    END DO
    DO K=1,N3
      DO I=II,MIN0(II+NB-1,N1)
        X(I,J,K)=ZWORK(K,I-II+1)
      END DO
    END DO
  END DO
END DO
```

```
DO K=1,N3
  DO II=1,N1,NB
    DO JJ=1,N2,NB
      DO I=II,MIN0(II+NB-1,N1)
        DO J=JJ,MIN0(JJ+NB-1,N2)
          YWORK(J,I-II+1)=X(I,J,K)
        END DO
      END DO
    END DO
    DO I=II,MIN0(II+NB-1,N1)
      CALL IN_CACHE_FFT(YWORK(1,I-II+1),N2)
    END DO
    DO J=1,N2
      DO I=II,MIN0(II+NB-1,N1)
        X(I,J,K)=YWORK(J,I-II+1)
      END DO
    END DO
  END DO
  DO J=1,N2
    CALL IN_CACHE_FFT(X(1,J,K),N1)
  END DO
END DO
```

Fig. 3. A block three-dimensional FFT algorithm

to the global index as the *cyclic* distribution:

$$J_r = \hat{J}_r(m) \times P + m, \quad 0 \le m \le P-1, \quad 1 \le r \le 3. \tag{4.2}$$

To illustrate the all-to-all communication it is convenient to decompose N_i into two dimensions, $\tilde{N}_i$ and P_i, where $\tilde{N}_i \equiv N_i/P_i$. Although P_i is the same as P, we are using the subscript i to indicate that this index belongs to dimension i.

Starting with the initial data $\hat{x}(\hat{N}_1,\, N_2,\, N_3)$, the parallel three-dimensional FFT can be performed according to the following steps:

Step 1: Transpose

$$\hat{x}_1(J_3,\, \hat{J}_1,\, J_2) = \hat{x}(\hat{J}_1,\, J_2,\, J_3).$$

Step 2: $(N_1/P) \cdot N_2$ individual N_3-point multicolumn FFTs

$$\hat{x}_2(K_3,\, \hat{J}_1,\, J_2) = \sum_{J_3=0}^{N_3-1} \hat{x}_1(J_3,\, \hat{J}_1,\, J_2)\omega_{N_3}^{J_3K_3}.$$

Step 3: Rearrangement

$$\begin{aligned}\hat{x}_3(\hat{J}_1,\, J_2,\, \tilde{K}_3,\, P_3) &= \hat{x}_2(P_3,\, \tilde{K}_3,\, \hat{J}_1,\, J_2)\\ &\equiv \hat{x}_2(K_3,\, \hat{J}_1,\, J_2).\end{aligned}$$

Step 4: All-to-all communication

$$\hat{x}_4(\tilde{J}_1,\, J_2,\, \hat{K}_3,\, P_1) = \hat{x}_3(\hat{J}_1,\, J_2,\, \tilde{K}_3,\, P_3).$$

Step 5: Rearrangement

$$\hat{x}_5(J_2,\, \tilde{J}_1,\, \hat{K}_3,\, P_1) = \hat{x}_4(\tilde{J}_1,\, J_2,\, \hat{K}_3,\, P_1).$$

Step 6: $N_1 \cdot (N_3/P)$ individual N_2-point multicolumn FFTs

$$\hat{x}_6(K_2,\, \tilde{J}_1,\, \hat{K}_3,\, P_1) = \sum_{J_2=0}^{N_2-1} \hat{x}_5(J_2,\, \tilde{J}_1,\, \hat{K}_3,\, P_1)\omega_{N_2}^{J_2K_2}.$$

Step 7: Rearrangement

$$\begin{aligned}\hat{x_7}(J_1, K_2, \hat{K_3}) &\equiv \hat{x_7}(P_1, \tilde{J}_1, K_2, \hat{K_3}) \\ &= \hat{x_6}(K_2, \tilde{J}_1, \hat{K_3}, P_1).\end{aligned}$$

Step 8: $N_2 \cdot (N_3/P)$ individual N_1-point multicolumn FFTs

$$\hat{y}(K_1, K_2, \hat{K_3}) = \sum_{J_1=0}^{N_1-1} \hat{x_7}(J_1, K_2, \hat{K_3})\omega_{N_1}^{J_1K_1}.$$

The distinctive features of the parallel three-dimensional FFT algorithm can be summarized as:

- The parallel three-dimensional FFT is accompanied with a local transpose (data rearrangement).
- $N^{2/3}/P$ individual $N^{1/3}$-point multicolumn FFTs are performed in steps 2, 6 and 8 for the case of $N_1 = N_2 = N_3 = N^{1/3}$.
- The all-to-all communication occurs just once.

If both of N_1 and N_3 are divisible by P, the workload on each processor is also uniform.

Although the input data $\hat{x}(\hat{N_1}, N_2, N_3)$ is distributed along the first dimension N_1, the output data $\hat{y}(N_1, N_2, \hat{N_3})$ is distributed along the third dimension N_3. If we assume that the input data and output data are both the same distribution, an additional one all-to-all communication step is needed.

5 In-cache FFT Algorithm

We use the radix-2, 4 and 8 Stockham autosort algorithm [18] for in-cache FFT.

Although the Stockham autosort algorithm requires a scratch array the same size as the input data array, it is unnecessary the digit-reverse permutation. If the Stockham autosort algorithm is used for the individual FFTs, the additional scratch requirement for performing the individual FFTs is $O(N^{1/3})$ (where $N = N_1 \times N_2 \times N_3$) at most.

The higher radices are more efficient in terms of both memory and floating-point operations. A high ratio of floating-point instructions to memory operations is particularly important in a cache-based processor. In view of the high ratio of floating-point instructions to memory operations, the radix-8 FFT is more advantageous than the radix-4 FFT. A power-of-two FFT (except for 2-point FFT) can be performed by a combination of radix-8 and radix-4 steps containing at most two radix-4 steps. That is, the power-of-two FFTs can be performed as a length $n = 2^p = 4^q 8^r$ $(p \geq 2,\ 0 \leq q \leq 2,\ r \geq 0)$.

6 Performance Results

To evaluate the implemented parallel three-dimensional FFT, named FFTE[1] (version 3.2), we compared its performance against that of the FFTW library (version 2.1.5) [15],

[1] `http://www.ffte.jp`

Table 1. Performance of parallel three-dimensional FFTs on dual Xeon PC SMP cluster

P (Nodes × CPUs)	$N_1 \times N_2 \times N_3$	FFTE 3.2 (SSE2)		FFTE 3.2 (x87)		FFTW 2.1.5	
		Time	MFLOPS	Time	MFLOPS	Time	MFLOPS
1×1	$2^8 \times 2^8 \times 2^8$	7.68317	262.04	8.33619	241.51	12.86992	156.43
1×2	$2^8 \times 2^8 \times 2^8$	4.29060	469.23	4.38417	459.21	10.77560	186.84
2×1	$2^8 \times 2^8 \times 2^9$	6.31334	664.36	6.76740	619.78	16.21513	258.67
2×2	$2^8 \times 2^8 \times 2^9$	5.40536	775.95	5.58136	751.48	11.89469	352.62
4×1	$2^8 \times 2^9 \times 2^9$	7.91637	1102.04	8.31000	1049.84	17.43453	500.40
4×2	$2^8 \times 2^9 \times 2^9$	6.61002	1319.84	6.75353	1291.79	13.59440	641.75
8×1	$2^9 \times 2^9 \times 2^9$	8.80932	2056.84	9.38208	1931.28	17.98921	1007.24
8×2	$2^9 \times 2^9 \times 2^9$	7.21630	2510.90	7.48793	2419.81	13.78780	1314.16
16×1	$2^9 \times 2^9 \times 2^{10}$	9.35400	4017.64	10.28623	3653.52	18.42153	2040.06
16×2	$2^9 \times 2^9 \times 2^{10}$	7.45187	5043.16	8.02600	4682.40	14.56551	2580.13

which is known as one of the fastest FFT libraries for many processors. Although the latest FFTW (version 3.0.1) supports
SSE/SSE2/3DNow!/AltiVec (new in version 3.0), MPI parallel transforms are still only available in 2.1.5.

We averaged the elapsed times obtained from 10 executions of complex forward FFTs. The parallel FFTs were performed on double-precision complex data, and the table for twiddle factors was prepared in advance. We used transposed order output to reduce the all-to-all communication step for the parallel block three-dimensional FFT and the FFTW.

A 16-node dual Xeon PC SMP cluster (Prestonia 2.8 GHz, 12 K uops L1 instruction cache, 8 KB L1 data cache, 512 KB L2 cache, 2 GB DDR-SDRAM main memory per node, Linux 2.4.20smp) was used. The nodes on the PC SMP cluster are interconnected through a Myrinet-2000 switch.

MPICH-SCore [19] was used as a communication library. We used an intranode MPI library for the PC SMP cluster.

The Intel C++ Compiler (`icc`, version 7.0) and the Intel Fortran Compiler (`ifc`, version 7.0) were used on the dual Intel Xeon PC cluster. For the FFTE (SSE2), the compiler options used were specified as "`icc -O3 -xW -fno-alias`" and "`ifc -O3 -xW -fno-alias`." For the FFTE (x87), the compiler options used were specified as "`ifc -O3 -fno-alias`." For the FFTW, the compiler options used were specified as "`icc -O3 -xW`" and "`ifc -O3 -xW`."

Table 1 compares the FFTE (SSE2 and x87) and the FFTW in terms of their run times and MFLOPS. The first column of the table indicates the number of processors. The second column gives the problem size. The next six columns contain the average elapsed time in seconds and the average execution performance in MFLOPS. The MFLOPS values are each based on $5N \log_2 N$ for a transform of size $N = 2^m$.

Table 2 shows the results of the all-to-all communication timings on the dual Xeon PC SMP cluster. The first column of the table indicates the number of processors. The

Table 2. All-to-all communication performance on dual Xeon PC SMP cluster

P	$N_1 \times N_2 \times N_3$	Time	MB/sec
1×2	$2^8\times2^8\times2^8$	0.73685	91.08
2×1	$2^8\times2^8\times2^9$	1.44647	92.79
2×2	$2^8\times2^8\times2^9$	1.38669	72.59
4×1	$2^8\times2^9\times2^9$	2.36666	85.07
4×2	$2^8\times2^9\times2^9$	2.28441	51.41
8×1	$2^9\times2^9\times2^9$	3.05918	76.78
8×2	$2^9\times2^9\times2^9$	2.85692	44.04
16×1	$2^9\times2^9\times2^{10}$	3.37355	74.60
16×2	$2^9\times2^9\times2^{10}$	3.11048	41.80

second column gives the problem size. The next two columns contain the average elapsed time in seconds and the average bandwidth in MB/sec.

In Tables 1 and 2, we can clearly see that all-to-all communication overhead contributes significantly to the execution time. For this reason, the difference in performance between the implemented parallel three-dimensional FFT and the FFTW decreases according to increasing the number of processors.

For $N = 2^9\times2^9\times2^{10}$ and $P = 16\times2$, FFTE (SSE2) runs about 1.95 times faster than the FFTW, as shown in Table 1.

The performance of the implemented parallel three-dimensional FFT remains at a high level, even for the larger problem size, because of cache blocking. Moreover, our implementation of the parallel three-dimensional FFT exploits the SSE2 instructions. These are two reasons why the implemented parallel three-dimensional FFT is the most advantageous with the dual Xeon PC SMP cluster.

These results clearly indicate that the implemented FFT is superior to the FFTW.

We note that on a 16-node dual Xeon 2.8 GHz PC SMP cluster, over 5 GFLOPS was realized with size $N = 2^9 \times 2^9 \times 2^{10}$ in the FFTE, as shown in Table 1.

7 Conclusion

In this paper, we proposed the implementation of the parallel three-dimensional FFT using short vector SIMD instructions on clusters of PCs. We vectorized FFT kernels using the SSE2 instructions, and parallelized the block three-dimensional FFT.

The performance of the implemented parallel three-dimensional FFT remains at a high level, even for a larger problem size, because of cache blocking.

These results demonstrate that the implemented FFT utilizes cache memory effectively. We succeeded in obtaining a performance of over 5 GFLOPS on a 16-node dual Xeon 2.8 GHz PC SMP cluster. These performance results demonstrate that the proposed parallel block three-dimensional FFT algorithm utilizes cache memory effectively.

References

1. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19** (1965) 297–301
2. Brass, A., Pawley, G.S.: Two and three dimensional FFTs on highly parallel computers. Parallel Computing **3** (1986) 167–184
3. Agarwal, R.C., Gustavson, F.G., Zubair, M.: An efficient parallel algorithm for the 3-D FFT NAS parallel benchmark. In: Proceedings of the Scalable High-Performance Computing Conference. (1994) 129–133
4. Hegland, M.: Real and complex fast Fourier transforms on the Fujitsu VPP 500. Parallel Computing **22** (1996) 539–553
5. Calvin, C.: Implementation of parallel FFT algorithms on distributed memory machines with a minimum overhead of communication. Parallel Computing **22** (1996) 1255–1279
6. Takahashi, D.: Efficient implementation of parallel three-dimensional FFT on clusters of PCs. Computer Physics Communications **152** (2003) 144–150
7. Nadehara, K., Miyazaki, T., Kuroda, I.: Radix-4 FFT implementation using SIMD multimedia instructions. In: Proc. 1999 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '99). Volume 4. (1999) 2131–2134
8. Franchetti, F., Karner, H., Kral, S., Ueberhuber, C.W.: Architecture independent short vector FFTs. In: Proc. 2001 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001). Volume 2. (2001) 1109–1112
9. Rodriguez V, P.: A radix-2 FFT algorithm for modern single instruction multiple data (SIMD) architectures. In: Proc. 2002 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2002). Volume 3. (2002) 3220–3223
10. Kral, S., Franchetti, F., Lorenz, J., Ueberhuber, C.W.: SIMD vectorization of straight line FFT code. In: Proc. 9th International Euro-Par Conference (Euro-Par 2003). Volume 2790 of Lecture Notes in Computer Science., Springer-Verlag (2003) 251–260
11. Frigo, M., Johnson, S.G.: The design and implementation of FFTW3. In: Proc. IEEE. Volume 93. (2005) 216–231
12. Franchetti, F., Kral, S., Lorenz, J., Ueberhuber, C.W.: Efficient utilization of SIMD extensions. In: Proc. IEEE. Volume 93. (2005) 409–425
13. Bailey, D.H.: FFTs in external or hierarchical memory. The Journal of Supercomputing **4** (1990) 23–35
14. Van Loan, C.: Computational Frameworks for the Fast Fourier Transform. SIAM Press, Philadelphia, PA (1992)
15. Frigo, M., Johnson, S.G.: FFTW: An adaptive software architecture for the FFT. In: Proc. 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 98). (1998) 1381–1384
16. Intel Corporation: IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture. (2004)
17. Intel Corporation: Intel C++ Compiler for Linux Systems User's Guide. (2004)
18. Swarztrauber, P.N.: FFT algorithms for vector computers. Parallel Computing **1** (1984) 45–63
19. Sumimoto, S., Tezuka, H., Hori, A., Harada, H., Takahashi, T., Ishikawa, Y.: High performance communication using a commodity network for cluster systems. In: Proc. Ninth International Symposium on High Performance Distributed Computing (HPDC-9). (2000) 139–146

Other Para'04 Contributed Talks

The contributed talks below can be found in the following report:

- J. Dongarra, K. Madsen and J. Waśniewski (Eds.)
  - ▶ Complementary proceedings of the Para'04 Workshop on State-of-the-Art in Scientific Computing, Lyngby, Denmark, June, 2004.
  - ▶ IMM-Technical report-2005-09.
  - ▶ Informatics and Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, Denmark.
  - ▶ URL: http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=3927

- **Masking Latency with Data Driven Program Variants** by Scott B. Baden.
- **Dynamic Code Generation and Component Composition in C++ for Optimising Scientific Codes at Run-time** by Olav Beckmann and Paul H J Kelly.
- **Communication Strategies for Parallel Cooperative Ant Colony Optimization on Clusters and Grids** by Siegfried Benkner, Karl F. Doerner, Richard F. Hartl, Guenter Kiechle and Maria Lucka.
- **Fully Self Organized Public Key Management for Mobile Ad Hoc Network** by Daeseon Choi, Seunghun Jin, and Hyunsoo Yoon.
- **Predicting Protein-Protein Interactions in Parallel** by Yoojin Chung, Sang-Young Cho and Chul-Hwan Kim
- **On the Parallelization of the Lattice-Boltzmann Method** by Salvatore Filippone, Nicola Rossi, Gino Bella and Stefano Ubertini.
- **Parallel and Distributed Techniques for Extracting Large Ontologies as a Resource in a Grid Environment** by Andrew Flahive, Mehul Bhatt, Carlo Wouters, Wenny Rahayu, David Taniar, and Tharam Dillon.
- **Adaptive Fuzzy Active Queue Management** by Mahdi Jalili-Kharaajoo, Mohammadreza Sadri and Farzad Habibipour Roudsari.
- **Inversion and Division Architecture in Elliptic Curve Cryptography over** $GF(2^n)$ by Jun-Cheol Jeon, Kyo-Min Ku, and Kee-Young Yoo.
- **Efficient On-the-fly Detection of First Races in Nested Parallel Programs** by Keum-Sook Ha, Yong-Kee Jun, and Kee-Young Yoo.
- **New Architecture for Inversion and Division over GF(2^m)** by Kyeoung Ju Ha, Kyo Min Ku, and Kee Young Yoo.
- **Solving Linear Systems on Cluster Computers with High Accuracy** by Carlos Amaral Hölbig, Paulo Sérgio Morandi Jr., Bernardo Frederes Krämer Alcalde, Tiarajú Asmuz Diverio, and Dalcidio Moraes Claudio.
- **Finite Fields Multiplier based on Cellular by Automata** by Hyun-Sung Kim, and Il-Soo Jeon.
- **Efficient Systolic Architecture for Modular Multiplication over GF(2^m)** by Hyun-Sung Kim and Sung-Woon Lee.

- **Analyzing the Safety Problem in Securitay Systems using SPR Tool** by Il-Gon Kim, Jin-Young Choi, Peter D. Zegzhda, Maxim O. Kalinin, Dmitry P. Zegzhda, In-Hye Kang, Pil-Yong Kang and Wan S. Yi.
- **A CBD-based SSL Component Model** by Young-Gab Kim, Lee-Sub Lee, Dong-won Jeong, Young-Shil Kim, and Doo-Kwon Baik.
- **Exponentiation over GF(2^m) For Public Key Crypto System using Cellular Automata** by Kyo Min Ku, Kyeoung Ju Ha, and Kee Young Yoo.
- **Area Efficient Multiplier based on LFSR Architecture** by Jin-Ho Lee and Hyun-Sung Kim.
- **Efficient Authentication and Key Agreement for Client-Server Environment** by Sung-Woon Lee, Hyun-Sung Kim, and Kee-Young Yoo.
- **New Efficient Digit-Serial Systolic AB^2 Multiplier & Divider in $GF(2^m)$** by Won-Ho Lee and Kee-Young Yoo.
- **Parallel Co-Processor for Ultra-fast Line Drawing** by Pere Marès, Antonio B. Martínez and Joan Aranda.
- **The Design Patterns of Performance-Decision Factors in High-Speed NIDS** by Jongwoon Park, Keewan Hong, Kiyoong Hong, Dongkyoo Kim, and Bongnam No.
- **Scalable Race Visualization for Debugging Message-Passing Programs** by Mi-Young Park, So-Hee Park, Su-Yun Bae, and Yong-Kee Jun.
- **Hierarchical Structures for Multi-Resolution Visualization of AMR Data** by Sanghun Park.
- **The Development of Domain Specific Languages From Scientific Libraries** by Daniel Quinlan.
- **A method to Derive the Cache Performance of Irregular Applications on machines with Direct Mapped Caches** by Carsten Scholtes.
- **Interfacing C++ member functions with C libraries** by Kurt Vanmechelen and Jan Broeckhove.
- **Compilation Techniques for a Chip-Multiprocessor with Two Execution Modes** by Chao-Chin Wu.
- **Explicit Formulas and Library of Images of Electromagnetic Fields for Anisotropic Materials** by Valery Yakhno, Tatyana Yakhno and Mustafa Kasap.

Author Index