José M. Laginha M. Palma
Patrick R. Amestoy Michel Daydé
Marta Mattoso João Correia Lopes (Eds.)

High Performance Computing for Computational Science – VECPAR 2008

LNCS 5336

8th International Conference
Toulouse, France, June 2008
Revised Selected Papers

Springer

Lecture Notes in Computer Science 5336

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

José M. Laginha M. Palma
Patrick R. Amestoy Michel Daydé
Marta Mattoso João Correia Lopes (Eds.)

High Performance Computing for Computational Science - VECPAR 2008

8th International Conference
Toulouse, France, June 24 -27, 2008
Revised Selected Papers

Springer

Volume Editors

José M.Laginha M. Palma
João Correia Lopes
University of Porto, Faculty of Engineering
4200-465 Porto Portugal
E-mail: {jpalma, jlopes}@fe.up.pt

Patrick R. Amestoy
Michel Daydé
University of Toulouse, INP (ENSEEIHT), IRIT
31071 Toulouse CEDEX 7, France
E-mail: {amestoy, michel.dayde}@enseeiht.fr

Marta Mattoso
Federal University of Rio de Janeiro
21941-972 Rio de Janeiro RJ, Brazil
E-mail: marta@cos.ufrj.br

Library of Congress Control Number: Applied for

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

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

ISSN 0302-9743
ISBN-10 3-540-92858-8 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-92858-4 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12591596 06/3180 5 4 3 2 1 0

Preface

VECPAR is an international conference series dedicated to the promotion and advancement of all aspects of high-performance computing for computational science, as an industrial technique and academic discipline, extending the frontier of both the state of the art and the state of practice. The audience and participants of VECPAR are researchers in academic departments, government laboratories and industrial organizations. There is now a permanent website for the conference series in `http://vecpar.fe.up.pt` where the history of the conference is described.

The 8th edition of VECPAR was organized in Toulouse (France), June 24–27, 2008. It was the third time the conference was celebrated outside Porto after Valencia (Spain) in 2004 and Rio de Janeiro (Brazil) in 2006.

The conference programme consisted of 6 invited talks and 53 accepted papers out of 73 contributions that were initially submitted.

The major themes are divided into:

- Large-Scale Simulations and Numerical Algorithms in Computer Science and Engineering (aerospace, earth, environment, finance, geoscience)
- Computing in Healthcare and Biosciences
- Multiscale and Multiphysics Problems
- Cluster Computing and Grid Computing
- Data-Centric and High-Productivity Computing
- Cooperative Engineering
- Problem-Solving Environments
- Imaging and Graphics

Two workshops, in addition to tutorials, were organized before the conference:

HPDGrid 2008 — International Workshop on High-Performance Data Management in Grid Environments on June 24 and Sparse Days at CERFACS on June 23 and 24.

The most significant contributions are made available in the present book, edited after the conference and after a second review of all orally presented papers at VECPAR 2008 and at the workshops.

Note also that the conference gave the opportunity of highlighting French–Japanese cooperations in grid computing and computational science with a session devoted to the REDIMPS Project within the VECPAR conference track and two workshops organized on June 23 and 24: PAAP[1] and NEGST[2].

Melissa Paes received the Best Student Presentation award for her talk on "High-Performance Query Processing" (see corresponding paper in Session Data-Centric and High-Productivity Computing).

[1] `http://www2.lifl.fr/MAP/paap/`
[2] `http://www2.lifl.fr/MAP/negst/`

The eighth edition of VECPAR took place in Toulouse at INPT/ENSEEIHT: a school of engineers located in the center of Toulouse.

The organizational aspects were dealt with by colleagues from the "Institut de Recherche en Informatique de Toulouse" (IRIT) with the participation of colleagues from CERFACS and the support of the Institut National Polytechnique de Toulouse / ENSEEIHT, Université Paul Sabatier, Pôle de Compétivité "Aerospace Valle" and Région Midi-Pyrénées.

Paper submission and selection were managed via the conference management system, held and operated by the Faculty of Engineering of the University of Porto (FEUP). Websites were maintained both by FEUP and IRIT. Registrations were managed by IRIT.

The success of the VECPAR conferences and its long life are a result of the collaboration of many people. As before, given the widespread organization, a large number of collaborators were involved.

We mention only some, and through them we thank many others who offered most of their time and commitment to the success of the conference and workshops:

- Véronique Debats (IRIT)
- Sabyne Lartigues (IRIT)
- Sylvie Soler (INPT/ENSEEIHT)

For their contribution to the present book, we must thank all the authors for meeting the deadlines and all members of the Scientific Committee who helped us so much in selecting the papers. We also thanks the members of the committees involved in the organization of the workshops held before the conference.

October 2008

José M.L.M. Palma
Patrick Amestoy
Michel Daydé
Marta Mattoso
João Correia Lopes

VECPAR is a series of conferences organized by the Faculty of Engineering of Porto (FEUP) since 1993.

Organization

Organizing Committee

M. Daydé	IRIT/INPT - ENSEEIHT, France (Chair)
L. Giraud	IRIT/INPT - ENSEEIHT, France
R. Guivarch	IRIT/INPT - ENSEEIHT, France
D. Hagimont	IRIT/INPT - ENSEEIHT, France
J.-P. Jessel	IRIT/UPS, France
J. Correia Lopes	FEUP/INESC Porto, Portugal (Web Chair)

Steering Committee

J. Palma	Portugal (Chair)
J. Dongarra	USA
A. Coutinho	Brazil
J. Fortes	USA
V. Hernandez	Spain
O. Marques	USA
K. Miura	Japan

Scientific Committee

P. Amestoy	France (Chair)
Jacques Bahi	France
Carlos Balsa	Portugal
Rob Bisseling	The Netherlands
Vincent Breton	France
Xiao-Chuan Cai	USA
Mark A. Christon	USA
Olivier Coulaud	France
José Cardoso e Cunha	Portugal
Rudnei Cunha	Brazil
Michel J. Daydé	France
Frédéric Desprez	France
Philippe Devloo	Brazil
Marcos Donato Ferreira	Brazil
Jack Dongarra	USA
Inês de Castro Dutra	Portugal
Renato Figueiredo	USA
Wolfgang Gentzsch	Germany
Alan D. George	USA
Omar Ghattas	USA

Luc Giraud	France
Serge Gratton	France
Ronan Guivarch	France
Daniel Hagimont	France
Abdelkader Hameurlain	France
Bruce Hendrickson	USA
Thomas Hérault	France
Vicente Hernandez	Spain
Vincent Heuveline	Germany
Jean-Pierre Jessel	France
Christopher R. Johnson	USA
Jacko Koster	Norway
Dieter Kranzmueller	Austria
Stéphane Lanteri	France
Xiaoye Sherry Li	USA
Kuan-Ching Li	Taiwan
Rainald Lohner	USA
Thomas Ludwig	Germany
Osni Marques	USA
Armando Jorge Padilha	Portugal
José Laginha Palma	Portugal
Dennis Parsons	USA
Serge G. Petiton	France
Thierry Priol	France
Heather Ruskin	Ireland
Mitsuhisa Sato	Japan
Satoshi Sekiguchi	Japan
António Augusto Sousa	Portugal
Mark A. Stadtherr	USA
Domenico Talia	Italy
Francisco Tirado	Spain
Miroslav Tuma	Czech Republic
Patrick Valduriez	France
Xavier Vasseur	France
Roland Wismuller	Germany

Invited Speakers

Nicholas Ayache	INRIA, France
Philippe Bougeault	European Centre for Medium-Range Weather Forecasts, UK
Jack Dongarra	University of Tennessee, USA
Satoshi Matsuoka	Tokyo Institute of Technology, Japan
Thierry Poinsot	CERFACS, France
Michael A. Puso	Lawrence Livermore National Laboratory, USA

Additional Reviewers

Jose Miguel Alonso
Fernando Alvarruiz
Gabriel Antoniu
Eugenio Cesario
Laurent Choy
Grzegorz Cieslewski
Olivier Coulaud
A. Esnard
P. Henon
Benoit Hudzia
Martin Quinson
Yvon Jégou
G. Latu
Kuan-Ching Li
Rodrigo F. de Mello
Christian Perez
Giuseppe Pirrò
Jose E. Roman
Hung-Hsun Su
Christian Tenllado
Tien-Hsiung Weng

Sponsoring Organizations

The Organizing Committee is very grateful to the following organizations for their support:

AESE	Pôle de Compétivité "Aerospace Valley"
CERFACS	European Centre for Research and Advanced Training in ENSEEIHT
INPT	Institut National Polytechmique de Toulouse
IRIT	Institut de Recherche en Informatique de Toulouse Région Midi-Pyrénées
UP	Universidade do Porto, Portugal
UPS	Université Paul Sabatier

HPDgrid 2008
International Workshop on High-Performance Data Management in Grid Environments

Initially developed for the scientific community as a generalization of cluster computing using the Web, grid computing is gaining much interest in other important areas such as enterprise information systems. This makes data management more critical than ever. Compared with cluster computing which deals with homogeneous parallel systems, grids are characterized by high heterogeneity, high autonomy and large-scale distribution of computing and data resources. Managing and transparently accessing large numbers of autonomous, heterogeneous data resources efficiently is an open problem. Furthermore, different grids may have different requirements with respect to autonomy, query expressiveness, efficiency, quality of service, fault-tolerance, security, etc. Thus, different solutions need be investigated, ranging from extensions of distributed and parallel computing techniques to more decentralized, self-adaptive techniques such as peer-to-peer (P2P).

The objective of the second edition of this one-day workshop was to bring together researchers and practitioners from the high-performance computing, scientific computing, distributed systems and database communities to discuss the challenges and propose novel solutions in the design and implementation of high-performance data management in Grid environments.

The following program was the result of the paper selection, with seven papers presented in two sessions: (1) data-intensive grid applications, and (2) replication in grids, grid services and grid data mining. In addition, we had two keynote sessions: the first one on "Grid Data Management: Open Problems and New Issues" by Esther Pacitti, and the second one on "Data Management in Pervasive Grids" by Jean-Marc Pierson. All selected papers were reviewed by three members of the Program Committee. We would like to thank them for the excellent work on coming up with a program that covers many different topics related to grid data management.

October 2008

Marta Mattoso
Alexandre Lima
Esther Pacitti
Patrick Valduriez

Organization

Workshop General Co-chairs

Esther Pacitti	INRIA and LINA, France
Alexandre A.B. Lima	Unigranrio, Brazil

Executive Chair

Patrick Valduriez	INRIA and LINA, France

Program Chair

Marta Mattoso	COPPE/UFRJ, Brazil

Program Committee

David Abramson	Monash University, Australia
Reza Akbarinia	University Nantes, France
Gabriel Antoniu	IRISA-INRIA, France
Rajkumar Buyya	University of Melbourne, Australia
Marco A. Casanova	PUC-Rio, Brazil
Inês de Castro Dutra	University Porto, Portugal
Alexandre Evsukoff	UFRJ, Brazil
Marcio Faerman	RNP, Brazil
Alvaro A.A. Fernandes	University Manchester, UK
Harald Kosch	University Passau, Germany
Nectarios Koziris	National Technical University Athens, Greece
Vidal Martins	PUC-Parana, Brazil
Wagner Meira Jr.	UFMG, Brazil
Philippe Navaux	UFRGS, Brazil
Jean-Marc Pierson	IRIT, France
Fabio Porto	EPFL, Switzerland
Gerson Sunye	University Nantes, France
Vladimir Vlassov	KTH, Sweden

Invited Speakers

Esther Pacitti	INRIA and LINA, Université de Nantes, France
Jean-Marc Pierson	IRIT, Université Paul Sabatier, France

Additional Reviewers

A.B.M. Russel
Alexandre Evsukoff
Álvaro A. Fernandes
George Tsoukalas
Rodrigo Virote Kassick
Eduardo Almeida

Table of Contents

Parallel and Distributed Computing

Cluster and Grid Computing

Problem Solving Environment and Data Centric

Numerical Methods

Linear Algebra

Computing in Geosciences and Biosciences

Imaging and Graphics

Computing for Aerospace and Engineering

High-Performance Data Management in Grid Environments

An Overview of High Performance Computing and Challenges for the Future

Jack Dongarra

University of Tennessee, USA

Abstract. In this talk we examine how high performance computing has changed over the last 10-year and look toward the future in terms of trends.

These changes have had and will continue to have a major impact on our software. A new generation of software libraries and algorithms are needed for the effective and reliable use of (wide area) dynamic, distributed and parallel environments.

Some of the software and algorithm challenges have already been encountered, such as management of communication and memory hierarchies through a combination of compile-time and run-time techniques, but the increased scale of computation, depth of memory hierarchies, range of latencies, and increased run-time environment variability will make these problems much harder.

We will focus on the redesign of software to fit multicore architectures.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, p. 1, 2008.

Parallelization of Sphere-Decoding Methods

Rafael A. Trujillo[1,2], Antonio M. Vidal[1], Víctor M. García[1], and Alberto González[3]

[1] Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, España
{rtrujillo,avidal,vmgarcia}@dsic.upv.es
[2] Departamento de Técnicas de Programación,
Universidad de las Ciencias Informáticas,
Carretera a San Antonio de los Baños Km 2 s/n, La Habana, Cuba
trujillo@uci.cu
[3] Departamento de Comunicaciones
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, España
agonzal@dcom.upv.es

Abstract. *Sphere-Decoding* (SD) methods are branch-and-bound-like techniques used for optimal detection of digital communications signals over in wireless MIMO (Multiple input Multiple Output) channels. These methods look for the optimal solution in a tree of partial solutions; the size of the tree depends on the parameters of the problem (dimension of the channel matrix, cardinality of the alphabet), and such search can be much more expensive depending on these parameters. This search often has to be carried out in real time. This paper presents parallel versions of the *Sphere-Decoding* method for different parallel architectures with the goal of reducing the computation time.

1 Introduction

Let us consider the following minimum squares problem:

$$\min_{s \in \mathcal{D}^m} \|x - Hs\|^2 \tag{1}$$

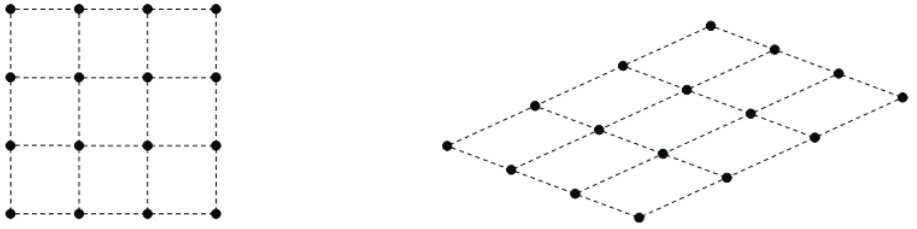
where $\mathcal{D}$ is a set of discrete values that can be finite or infinite. Since $\mathcal{D}$ is discrete, this can be considered as an Integer Programming problem.

The simplest method to solve this problem would be to solve the unconstrained minimum square problem

$$\min_{s} \|x - Hs\|^2 \tag{2}$$

and then "truncate" the solution to its closest value in $\mathcal{D}^m$. However, for many problems (specially when H is bad conditioned) this gives wrong results. Therefore, special methods have to be applied.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 2–12, 2008.



(a) (b)

Fig. 1. (a)Rectangular Lattice; (b)Skewed Lattice

Usually, this problem is described in terms of *Lattices*. Given $v_1, \cdots, v_m$ a set of linearly independent vectors, a lattice is the set of vectors

$$\lambda_1 v_1 + \cdots + \lambda_m v_m \quad \lambda_1, \cdots, \lambda_m \in \mathbb{Z} \tag{3}$$

If the elements of $\mathcal{D}$ are equally spaced (as is the case in communication problems) the set $\mathcal{D}^m$ forms a rectangular lattice like the displayed in Figure 1(a). When the elements of $\mathcal{D}^m$ are multiplied by the channel matrix H, they will form a skewed lattice (see Figure 1(b)).

The problem (1) is equivalent to the problem of finding the closest point (to a given point x) in a lattice with generating matrix H. This problem is known to be NP-Complete. It appears in the wireless communications field, where digital communications signals sent through systems with multiple send and receive antennas (*multiple input - multiple output* or *MIMO* systems) must be correctly "decoded" [4], [11].

The systems we are interested in, are composed of M transmit antennas and N receive antennas, through which a signal $\bar{s} = [\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_M]^T \in \mathbb{C}^M$ is transmitted. Real and imaginary parts of each component of the transmitted signal belong to a discrete set $\mathcal{D}$, finite ($|\mathcal{D}| = L$) and named *constellation* or *symbol alphabet*. The received signal $\bar{x} \in \mathbb{C}^N$ is a linear combination of the transmitted signal $\mathbf{s}$ plus a white gaussian noise term $\bar{v} \in \mathbb{C}^N$, with variance σ^2

$$\bar{x} = \bar{H}\bar{s} + \bar{v}. \tag{4}$$

Here, the channel matrix $\bar{H}$ is a general complex-valued matrix with N rows and M columns that models the MIMO channel response.

To simplify the programming details, usually the complex model (4) is transformed in a real model, where the vector s of dimension $m = 2M$, and the vectors x and v of dimensions $n = 2N$ are defined as:

$$s = \left[\mathcal{R}(\bar{s})^T \, \mathcal{I}(\bar{s})^T\right]^T, \; x = \left[\mathcal{R}(\bar{x})^T \, \mathcal{I}(\bar{x})^T\right]^T, \; v = \left[\mathcal{R}(\bar{v})^T \, \mathcal{I}(\bar{v})^T\right]^T,$$

and the matrix H of dimensions $n \times m$ as:

$$H = \begin{bmatrix} \mathcal{R}(\bar{H}) & \mathcal{I}(\bar{H}) \\ -\mathcal{I}(\bar{H}) & \mathcal{R}(\bar{H}) \end{bmatrix}$$

Thus, the real model equivalent to (4) is given by:

$$x = Hs + v. \tag{5}$$

The search within the whole lattice looking for the optimal solution of (1) where $|\mathcal{D}| = L$ would require an exponential time. However, there are better methods that take into account the special characteristic of problem (1). Examples of such methods are Kannan's algorithm [8], (where the search is limited to a rectangular parallelepiped), and the algorithms proposed by Fincke and Pohst [3] and improved by Schnorr and Euchner [9],[1] known as *sphere-decoding*, where the search is limited to an hyper-sphere of a given radius and with center in the point x.

In any case, *Sphere-Decoding* methods can be quite costly in time and memory when the problems grow in complexity; either by an increase in the number of transmitting-receiving antennas, by using larger alphabets, or by an increase in the variance of the noise.

In this paper several variants of the sphere-decoding algoritm are presented, for use in different parallel architectures. Other algorithms were developed, but, only those that gave better results are discussed.

A description of the *Sphere-Decoding* method is given in the next section, along with a formal description of the method following a *branch-and-bound* scheme. In the second section the parallelization of *Sphere-Decoding* for shared memory, distributed memory, and hybrid environments, is discussed. Finally, the experimental results and the conclusions will be presented.

2 Sphere-Decoding

The main idea in the *Sphere-Decoding* algorithm is to limit the search to the points in the lattice located into the sphere with center at the given vector x and radius r (see Fig. 2). Clearly, the closest point to x into the sphere shall be the closest point to x in the whole lattice. Since the search space is reduced, so is the computational complexity.

Mathematically speaking, SD algorithms find all the vectors $s \in \mathcal{D}^m$ such that

$$r^2 \geq \|x - Hs\|^2 \tag{6}$$

and then select the one minimizing the goal function. Using the QR decomposition of the matrix H and the orthogonality of the matrix Q, the condition (6) can be written as:

$$r^2 \geq \left\| x - \begin{bmatrix} Q_1 \; Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} s \right\|^2 = \left\| Q_1^T x - Rs \right\|^2 + \left\| Q_2^T x \right\|^2$$

or, equivalently:

$$r^2 - \left\| Q_2^T x \right\|^2 \geq \left\| Q_1^T x - Rs \right\|^2 \tag{7}$$

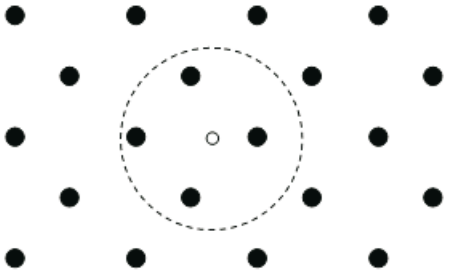


Fig. 2. Idea behind the sphere-decoding

Defining $y = Q_1^T x$ and $r'^2 = r^2 - \left\|Q_2^T x\right\|^2$ (7) can be rewritten as

$$r'^2 \geq \|y - Rs\|^2 \tag{8}$$

Since R is upper triangular, equation (8) can be written as

$$r'^2 \geq (y_m - R_{m,m}s_m)^2 + \|y_{1:m-1} - R_{1:m-1,1:m}s_{1:m}\|^2 \tag{9}$$

From this last condition, it can be deduced that a necessary condition for Hs to be located inside the sphere is that $r'^2 \geq (y_m - R_{m,m}s_m)^2$, or, equivalently, that the component s_m belongs to the interval

$$\left\lceil \frac{-r' + y_m}{R_{m,m}} \right\rceil \leq s_m \leq \left\lfloor \frac{r' + y_m}{R_{m,m}} \right\rfloor \tag{10}$$

where $\lceil \xi \rceil$ is the smallest element of the constellation greater or equal than ξ, and $\lfloor \xi \rfloor$ is the greatest element of the constellation smaller or equal than ξ. Therefore, for each value of s_m inside the interval (10), it is possible to determine the interval where the values of s_{m-1} will lie

$$\left\lceil \frac{-r'_{m-1} + y_{m-1|m}}{R_{m-1,m-1}} \right\rceil \leq s_{m-1} \leq \left\lfloor \frac{r'_{m-1} + y_{m-1|m}}{R_{m-1,m-1}} \right\rfloor \tag{11}$$

where $r'^2_{m-1} = r'^2 - (y_m - R_{m,m}s_m)^2$ and $y_{m-1|m} = y_{m-1} - R_{m-1,m}s_m$.

The algorithm continues following the same procedure to determine $s_{m-2}, s_{m-1}, \cdots, s_1$.

If no solution is found, the radius r must be increased and the algortihm must be executed again. A more detailed description of the *Sphere-Decoding* method, as well as an analysis of its computational complexity, can be found in [7].

The search in *Sphere-Decoding* belongs to the Branch–and–Bound general class; next, an algorithm implementing a general Branch-and-Bound search is displayed:

Algorithm 1. General Branch-and-Bound Algorithm to find the best solution

```
Variables:
S: LinearDataStructure;
N, solution : Node;
childs : ARRAY [1..MAXCHILDS] OF Node;
nChilds : INTEGER;
solutionValue : REAL;
 1: solution ← NoSolution(); solutionValue ← MAX
 2: Generate a initial node N_0
 3: Add(S, N_0) {Add N_0 to the Linear Data Structure S}
 4: while S is not empty do
 5:   N ← Extract(S) {A node of S is extracted and assigned to N}
 6:   [childs, nChilds] ← Branch(N);
 7:   for i = 1 to nChilds do
 8:     if IsAcceptable(childs[i]) then
 9:       if IsSolution(childs[i]) then
10:         if Value(childs[i]) < solutionValue then
11:           solutionValue ← Value(childs[i])
12:           solution ← childs[i]
13:         end if
14:       else
15:         Add(S, N) {Add the node N to the Linear Data Structure S}
16:       end if
17:     end if
18:   end for
19: end while
20: return solution
```

In the *Sphere-Decoding* method, the k-level nodes are the lattice points inside the sphere with radius r'_{m-k+1} and dimension $m-k+1$. The leaves of the tree would be the solutions of (6).

To fit the *Sphere-Decoding* method into a *Branch and Bound* scheme, each node should keep the following information:

- Tree level: $m-k+1$
- Value of $y_{k|k+1}$
- Value of r'_k
- Components of the vector $\hat{s}$ determined up to this moment: $\hat{s}_m, \ldots, \hat{s}_k$

where $k = m, m-1, \ldots, 1$. The generation of branches of a node of level $m-k+1$ (given in the algorithm 1 by the routine `Branch`) should generate as many nodes as elements has the alphabet or constellation, and the Bounding (given in the algorithm 1 by the routine `IsAcceptable`) should be carried out accepting only the nodes whose component $\hat{s}_{k-1}$ falls within the interval

$$\left[\left\lceil \frac{-r'_{k-1} + y_{k-1|k}}{R_{k-1,k-1}} \right\rceil, \left\lfloor \frac{r'_{k-1} + y_{k-1|k}}{R_{k-1,k-1}} \right\rfloor\right] \tag{12}$$

All nodes whose level is m are solution nodes. The routine `Value` is defined for these nodes, (points inside the hyper-sphere), which would return the value of $\|x - H\hat{s}\|^2$.

Since the *Sphere-Decoding* performs a depth-first search, looking for the best of all possible solutions, the nodes are stored using a LIFO (*Last-In First-Out*) Heap. This data structure is initialized using a special node N_0 whose level in the tree shall be 0, the value of $y_{m+1|m+2}$ shall be the last component of the vector y, the value of r_{m+1} might be the initial radius and the vector $\hat{s}$ would not have any defined component.

3 Sphere-Decoding Paralellization

In this section several possibilities for parallelization of the *Sphere-Decoding* method shall be discussed. The models considered shall be the shared memory model, distributed memory model and a hybrid model where several shared memory multiprocessors are interconnected. A study about parallelization of *Branch-and-Bound* methods can be found in [5], where different techniques are considered for distribution of workload among processors. In this section, it is described the adaptation of these techniques, for parallelization of *Sphere-Decoding*, using specific characteristics of the problem. including as well several novel proposals oriented to minimize the communications and to correctly balance the workload.

3.1 Distributed Memory Parallelization of Sphere-Decoding

The main issue in parallelizing *Sphere-Decoding* in a message-passing environment is the distribution of the tree among processors, so that the number of nodes (and the workload) is distributed as evenly as possible.

The parallel algorithm has the following structure: First, the *root* processor creates an initial node, and from this node starts a sequential execution of *Sphere-Decoding*, until enough nodes have been generated to be distributed among the rest of processors (at least one per processor). These nodes are distributed cyclically, so that each processor has its own data structure (the distribution is cyclic to avoid that some processors receive only nodes from the last levels, while others receive nodes from the first levels; the nodes from the last levels should generate much less work than the nodes from the first levels). See figure 3.

Then, in each iteration, each processor expands a fixed number of nodes in its structure. When these nodes are expanded, there will be a synchronization point so that all processors broadcast the state of its heap (empty or not) to the other processors. The processors whose heap is empty select randomly a processor to send a nodes *request* message. If it receives a negative (message *reject*) answer, it would choose another processor, until it receives a positive answer or has received a negative answer from all processors.

After expanding the pre-fixed number of nodes, the processors must check their queue of messages, to see whether they have some *request* message. In such

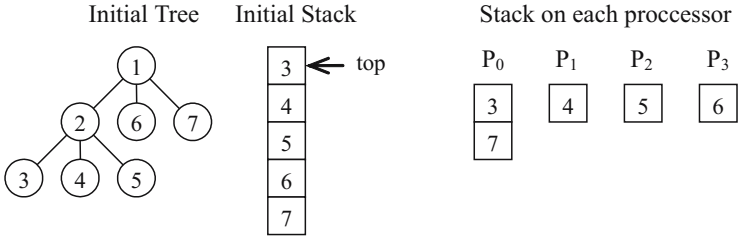


Fig. 3. Initial Expansion and distribution of nodes among processors

case, they answer with *reject* message if their structure is empty or *accept* if it has nodes to send. If a processor is going to send nodes to other processor, it will send a fixed percentage of nodes from its structure, chosen cyclically.

At the end of each iteration the processors must broadcast their best solution (to use it for comparison with solutions found in further iterations) and the state of their heap, to determine the end of the algorithm (which will happen when all heaps are empty).

An important issue is the choice of the number of nodes to expand by each processor. To avoid load imbalance, this number must be the same for each processor. This number should not be neither too small (it would increase the communications) nor too large (some processors might finish their search soon, and should wait a long time for the other processors to finish)

It is proposed by the authors that the estimation of the number of nodes to expand in a given iteration is carried out at the end of the former iteration; each processor estimates the number of nodes that might expand in the next iteration, broadcast the estimates to all processors, and the number of nodes to expand would be the median of all estimates.

The authors propose as well that each processor carries out its relative estimation taking into account the number of nodes in its heap, the levels of each node and the cardinal of the constellation $\mathcal{D}$. If the structure has l_1 nodes with level $m-1$ and l_2 nodes with level less than $m-1$, then the estimate of nodes that the processor might expand in the next iteration is

$$l_1 L + l_2 L^2 \tag{13}$$

In the parallel version of *Sphere-Decoding* implemented in this work, both proposals were included.

3.2 Shared Memory Parallelization of Sphere-Decoding

One of the drawbacks of the distributed memory parallelization is that, as shown above, it is necessary to fix the number of nodes to be expanded in each iteration, and check the state of the heap of each processor (empty or not). This number of nodes to expand must be carefully estimated, to avoid extra communications

or load imbalance. In a shared memory environment, this problem would not exist.

A first design for shared memory implementation would be that all processors share the heap, so that in each iteration each processor extracts a node, expands it and adds the generated new nodes to the heap. The algorithm would finish when the heap is empty.

The main problem of this design is that the operations of extraction and addition of nodes must be carried out in a critical section, that is, in a given time point only a single processor can add or extract nodes from the heap. It has been checked that this creates a severe bottleneck; therefore, to share the heap is not a good solution, and this means that each processor must keep and handle its own heap.

The algorithm proposed is, therefore, similar to the one described in the section 3.1. The main difference would be that there exist a global variable which controls whether any processor has an empty queue. This variable can be updated (within a critical section) by the first processor that becomes idle, and then by the other processors when a redistribution of the nodes is carried out. Therefore, all the processors search through their heaps until these become empty or until the global variable indicates that some processor finished its job. Then, all remaining nodes are collected in a global structure and redistributed cyclically over all processors.

This variant, which can be applied only in shared memory machines, decreases sensibly the likelihood of a workload imbalance.

3.3 Hybrid Parallelization of Sphere-Decoding

A different, hybrid, parallel architecture is formed by networks of shared memory multiprocessors (the memory of the global system is distributed, but the multiprocessors share a local memory). Of course, the message passing library MPI [10], can be used, although the message passing model may not be the most suitable to take full advantage of the architecture.

Keeping in mind the parallel SD algorithms for shared memory and for distributed memory, is easy to obtain an algorithm for the hybrid case. The algorithm would have the same parallel distribution that the distributed memory algorithm described in 3.1, while the search caried out in each multiprocessor would use the shared memory algorithm described in 3.2.

It must be considered that, in each iteration of the algorithm, each multiprocessor must select from its heap a fixed number of nodes to expand. Hence, the shared memory algorithm to be executed by the processors within a multiprocessor is slightly different from the algorithm described above. Apart from sharing a variable which indicates if a processor emptied its heap, they would share as well a variable with the number of nodes to be expanded in that iteration by the processor. Such variable should be decremented (in a critical region) by each processor every time that a node is expanded, and if this variable becomes zero all processors must stop to, through a reduction, determine the best solution found and rebuild the heap with the nodes not yet processed.

4 Experimental Results

The parallel algorithms outlined above where implemented in C, using the BLAS library [6] for the operations between vectors. The version for shared memory architecture was implemented using the OpenMP library [2], while the distributed memory version was build with MPI. Both libraries were used for the hybrid version. The algorithms were tested in a cluster composed of two PRIMERGY RXI600, each one with 4 Dual-Core 1.4 GHz Intel Itanium–2®processors, sharing a 4 GB RAM memory.

The experiments were designed so that the Sphere decoding algorithm generates a large number of nodes, and, consequently, the CPU time needed is also large. The sizes of the generated trees (total number of nodes) of the test problems were $5 \cdot 10^4$, $1 \cdot 10^5$, $5 \cdot 10^5$, $1 \cdot 10^6$, $4 \cdot 10^6$, $5 \cdot 10^6$, $8 \cdot 10^6$ and $1 \cdot 10^7$. These trees were generated in problems where $n = m = 8$, so that in all cases the number of levels of the tree is 8. It must be remarked that the execution time of the sphere decoding does not depend only on the number of nodes of the tree, but also on the number of possible solutions (last level nodes, which would represent lattice points inside the hyper-sphere). Given two trees with the same number of possible solutions, the one with smaller number of last-level nodes would need less execution time, since it would insert less nodes in the heap and there would be less nodes to expand.

Figure 4 shows the speedup of the MPI parallel version for the test problems considered. In the problems with smaller size (displayed with discontinuous lines) the speedup decreases for more than 4 processors. In the larger test cases there is a better speedup, although far from the optimum theoretical speedup.

The speedup for the OpenMP version is shown in Figure 5. This version was executed in one of the PRIMERGY RXI600 machines. It is quite remarkable

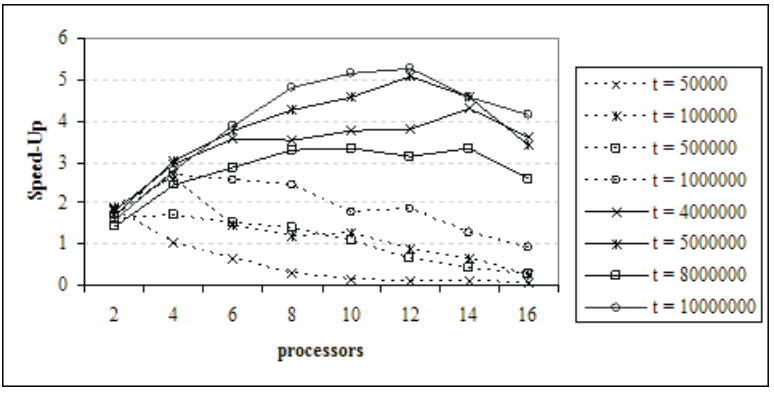


Fig. 4. SpeedUp of MPI parallel version of SD

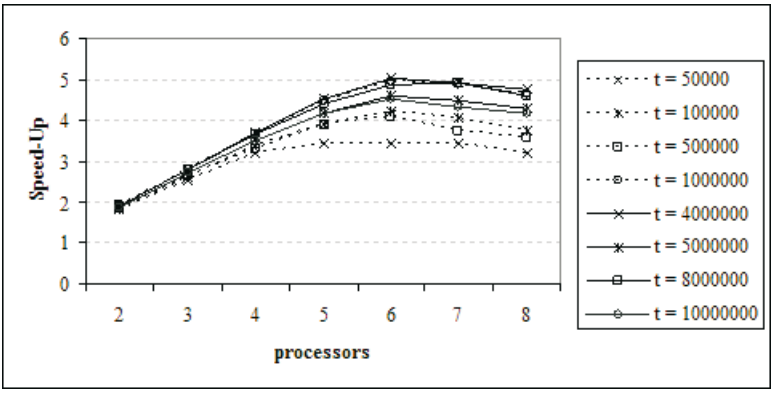


Fig. 5. SpeedUp of OpenMP parallel version of SD

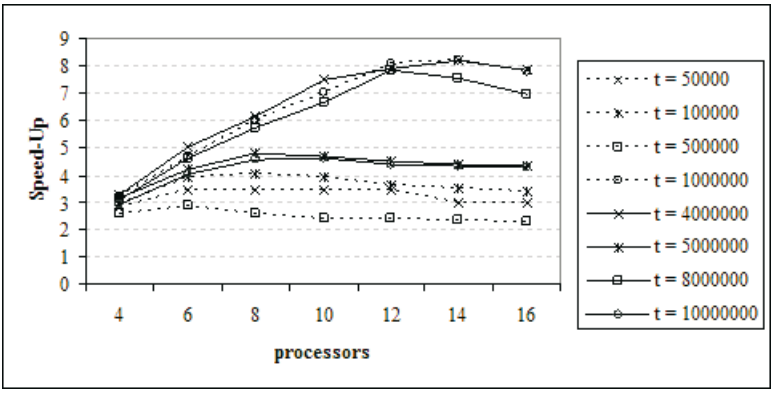


Fig. 6. SpeedUp of hybrid parallel version of SD

that all curves present the same trend, and in all problems the optimal number of processors is 6. The speedups in this case are close to theoretical optimum, better in most cases than the obtained with the MPI version.

Finally, the speedup of the hybrid version is displayed in Figure 6. This hybrid version was implemented in two levels, a global level implemented with MPI and a second, local level implemented with OpenMP. In some cases, the speedup obtained with this algorithm was far better than with the MPI or OpenMP versions. The problems in which the speedup was larger than 7 were those with a larger number of solution nodes in the tree (more than 70% of solution nodes). In problems where the percentage of solution nodes is small, the performance decreases, as in the case where the number of nodes is approximately $5 \cdot 10^5$. From these, only 11% were solution nodes.

5 Conclusions

Several possibilities for parallelization of the *Sphere-Decoding* method have been proposed. These cover three possible computing environments: Distribute memory, shared memory, and hybrid machines, where shared memory machines are connected. The hybrid version gave the greater speedups, although the shared memory version had more stable and consistent speedups. This was an expected behaviour, given the reduced communications and the possibility of a quite good work distribution. The workload balance is much more troublesome in distributed memory environments, so that the performance of the MPI and OpenMP+MPI versions suffers strong variations, depending on the structure of the tree generated during the search. The overall good performance of the OpenMP versions is quite relevant, since multicore processors and hybrid architectures are becoming increasingly popular.

Acknowledgements

This work was partially supported by the Vice-rectorate for Research, Development and Innovation of UPV within the programs: "Proyectos de Investigaciòn Multidisciplinares" (PAID-05-07, ref 6201) and "Primeros proyectos de Investigaciòn" (PAID-06-07, ref 20080009)

References

1. Agrell, E., Eriksson, T., Vardy, A., Zeger, K.: Closest point search in lattices. IEEE Transactions on Information Theory 48, 2201–2214 (2002)
2. Chandra, R., Dagun, L., Kohr, D., Maydan, D., McDonald, J.,, R.M.: Parallel Programming in OpenMP. Morgan Kaufmann Publishers, San Francisco (2000)
3. Fincke, U., Pohst, M.: Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. Mathematics of Computation 44(170), 463–471 (1985)
4. Foschini, G.J.: Layered space-time architecture for wireless communication in a fading environment when using multi-element antennas. Bell Labs Technical Journal 1, 41–59 (1996)
5. Grama, A., Gupta, A., Karypis, G., Kumar, V.: Introduction to Parallel Computing. Addison-Wesley, Reading (2003)
6. Hammarling, S., Dongarra, J., Du Croz, J., Hanson, R.J.: An extended set of fortran basic linear algebra subroutines. ACM Trans. Mat. Software (1988)
7. Hassibi, B., Vikalo, H.: On sphere decoding algorithm. i. expected complexity. IEEE Transactions on Signal Processing 53, 2806–2818 (2005)
8. Kannan, R.: Improved algorithms for integer programming and related lattice problems. ACM Symp. Theory of Computing (1983)
9. Schnorr, C.P., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Math. Programming 66, 181–191 (1994)
10. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: The Complete Reference. MIT Press, Cambridge (1996)
11. Telater, I.E.: Capacity of multi-antenna gaussian channels. Europ. Trans. Telecommun., 585–595 (1999)

Improving the Performance of a Verified Linear System Solver Using Optimized Libraries and Parallel Computation

Mariana Kolberg, Gerd Bohlender, and Dalcidio Claudio

Pontifícia Universidade Católica do Rio Grande do Sul, PPGCC
Av. Ipiranga, 6681 – 90619-900 – Porto Alegre, Brazil
Ph.: +55 51 3320-3611; Fax: +55 51 3320-3621
Universität Karlsruhe, Institut für Angewandte und Numerische Mathematik
Kaiserstr. 12 – 76128 – Karlsruhe, Germany
Ph.: +49 721 608 2049; Fax: +49 721 608 6679
{mkolberg,dalcidio}@inf.pucrs.br, bohlender@math.uka.de

Abstract. A parallel version of the self-verified method for solving linear systems was presented in [20, 21]. In this research we propose improvements aiming at a better performance. The idea is to implement an algorithm that uses technologies as MPI communication primitives associated to libraries as LAPACK, BLAS and C-XSC, aiming to provide both self-verification and speed-up at the same time. The algorithms should find an enclosure even for ill-conditioned problems. In this scenario, a parallel version of a self-verified solver for dense linear systems appears to be essential in order to solve bigger problems. Moreover, the major goal of this research is to provide a free, fast, reliable and accurate solver for dense linear systems.

Topics of the conference list: Numerical algorithms for CS&E, Parallel or Distributed Computation and Cluster Computation.

1 Introduction

Many real problems are simulated and modeled using dense linear systems of equations like $Ax = b$ with an $n \times n$ matrix $A \in \mathbb{R}^{n \times n}$ and a right hand side $b \in \mathbb{R}^n$. This is true for functional linear equations that occur like partial differential equations and integral equations that appear in several problems of Physics and Engineering [6]. Many different numerical algorithms contain this task as a subproblem.

There are numerous methods and algorithms which compute approximations to the solution x in floating-point arithmetic. However, usually it is not clear how good these approximations are, or if there exists a unique solution at all. In general, it is not possible to answer these questions with mathematical rigor if only floating-point approximations are used. These problems become especially difficult if the matrix A is ill conditioned. The use of self-verified methods can lead to more reliable results [14]. Verified computing provides an interval result that surely contains the correct result [19, 22]. Like that the algorithm also proves the existence and uniqueness of the solution of the problem.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 13–26, 2008.

The algorithm will, in general, succeed in finding an enclosure of the correct solution. If the solution is not found, the algorithm will let the user know.

The use of verified computing makes it possible to find the correct result. However, finding the verified result often increases the execution times dramatically [26]. The research already developed shows that the execution time of verified algorithms are much larger than the execution time of algorithms that do not use this concept [15, 16].

To compensate the lack of performance of such verified algorithms, some works suggest the use of midpoint-radius arithmetic to achieve a better performance, since its implementation can be done using only floating-point operations [28, 29]. This would be a way to increase the performance of verified methods.

The advent of parallel computing and its impact in the overall performance of many algorithms on numerical analysis can be seen in the past years [10]. The use of clusters plays an important role in such a scenario as one of the most effective manner to improve the computational power without increasing costs to prohibitive values. The parallel solutions for linear solvers found in the literature explore many aspects and constraints related to the adaptation of the numerical methods to high performance environments [7, 12, 13, 24, 27, 34]. However, those implementations do not deal with verified methods. In the field of verified computing many important contributions have been done in the last years. Some works related to verified solvers for dense linear systems [11, 14, 19, 25, 30] can be found in the literature. However, only a few papers on verified solvers for dense systems together with parallel implementations were found [18, 29, 33], but these authors implement other numerical problems or use a parallel approach for other architectures than clusters.

The new algorithms should find an enclosure even for very ill-conditioned problems. Moreover, the major goal of this research is to provide a free, fast, reliable and accurate solver for dense linear systems.

New algorithms based on the C-XSC (C for eXtended Scientific Computing) [19] methods were implemented, but using just libraries like BLAS (Basic Linear Algebra Subprograms [8]) and LAPACK [1, 23] to achieve better performance. The idea of reducing the switching of rounding mode presented by Bohlender was implemented as well as an optimization of the residuum based on the INTLAB [17] method. In other words, the new implementations try to join the best aspects of each library.

To ensure that an enclosure will be found, interval arithmetic was used. Several implementations and tests were done to find the most appropriate arithmetic to be used.

Aiming at a better performance, the algorithm was parallelized using the libraries SCALAPACK [3] (or Scalable LAPACK) and PBLAS. The idea of using popular and highly optimized libraries to gain performance will also be maintained in the parallel version.

One important advantage of the presented algorithm is the ability to find a solution even for very ill-conditioned problems (in tests on personal computers an enclosure could be found for condition number up to 10^{15}) while most algorithms may lead to an incorrect result when it is too ill-conditioned (above condition number 10^{8}). Our main contribution is to increase the use of verified computing through its optimization

and parallelization, once without parallel techniques it becomes the bottleneck of an application.

2 Verified Computing

The use of verified computing guarantee the mathematical rigor of the result. This is the most important advantage of such algorithms compared with ordinary methods. One possibility to implement verified computing is using interval arithmetic combined with suitable algorithms to find an interval result that will surely contain the correct result.

Section 2.1 presents some definitions of Interval Arithmetic. Considerations about the proprieties of a verified algorithm are discussed in Section 2.2.

2.1 Interval Arithmetic

Let $\mathbb{R}$ denote the set of real numbers and $\mathbb{PR}$ the power set over $\mathbb{R}$. The two most frequently used representations for intervals over $\mathbb{R}$, are the infimum-supremum representation

$$[a_1, a_2] := \{x \in \mathbb{R} : a_1 \leq x \leq a_2\} \text{ for some } a_1, a_2 \in \mathbb{R},\ a_1 \leq a_2, \tag{1}$$

where $\leq$ is the partial ordering $x \leq y$ and the midpoint-radius representation

$$\langle a, \alpha \rangle := \{x \in \mathbb{R} : |x - a| \leq \alpha\} \text{ for some } a \in \mathbb{R},\ 0 \leq \alpha \in \mathbb{R}. \tag{2}$$

The two representations are identical for real intervals (not for floating-point intervals), whereas for complex intervals the first representation are rectangles, the second one represents discs in the complex plane.

Today mostly the infimum-supremum arithmetic is used in computers. There are two main reasons for that. First, the standard definition of midpoint-radius arithmetic causes overestimation for multiplication and division, and second, the computed midpoint of the floating point result of an operation is, in general, not exactly representable in floating point, thus again causing overestimation and additional computational effort. However, in [29], Rump shows that the overestimation of operations using midpoint-radius representation compared to the result of the corresponding power set operation is limited by at most a factor 1.5 in radius.

In the computer implementation of interval arithmetic, special care has to be taken for the rounding [29]. Both infimum-supremum und midpoint-radius have advantages and disadvantages. Some intervals are better represented in one arithmetic and have an overestimation in other.

As presented in [29], the main point in using midpoint-radius arithmetic is that no case distinctions, switching of rounding mode in inner loops, etc. are necessary, only pure floating point matrix multiplications. And for those the fastest algorithms available may be used, for example, BLAS. The latter bear the striking advantages that

1. they are available for almost every computer hardware, and that
2. they are individually adapted and tuned for specific hardware and compiler configurations.

This gives an advantage in computational speed which is difficult to achieve by other implementations.

2.2 Suitable Algorithm

As mentioned before, we have to ensure that the mathematical properties of interval arithmetic as well as high accuracy arithmetic will be held if we want to achieve self-verification. Based on that, we used Algorithm 1 as the starting point of a first parallel version. This algorithm is based on the verified method fully described in [14] and will, in general, succeed in finding and enclosing a solution or, if it does not succeed, will let the user know. In the latter case, the user will know that the problem is likely to be very ill-conditioned or that the matrix A is singular. In this case, the user can try to use higher precision arithmetic.

Algorithm 1. Enclosure of a square linear system

1: $R \approx A^{-1}$ {Compute an approximate inverse using LU-Decomposition algorithm}
2: $\tilde{x} \approx R \cdot b$ {compute the approximation of the solution}
3: $[z] \supseteq R(b - A\tilde{x})$ {compute enclosure for the residuum}
4: $[C] \supseteq (I - RA)$ {compute enclosure for the iteration matrix}
5: $[w] := [z]$, $k := 0$ {initialize machine interval vector}
6: **while not** ($[w] \mathring{\subset} [y]$ or $k > 10$) **do**
7: $\quad [y] := [w]$
8: $\quad [w] := [z] + [C][y]$
9: $\quad k++$
10: **end while**
11: **if** $[w] \subseteq int[y]$ **then**
12: $\quad \Sigma(A, b) \subseteq \tilde{x} + [w]$ {The solution set (Σ) is contained in the solution found by the method}
13: **else**
14: $\quad$ no verification
15: **end if**

These enclosure method is based on the following interval Newton-like iteration:

$$x_{k+1} = Rb + (I - RA)x_k, k = 0, 1, ... \tag{3}$$

This equation is used to find a zero of $f(x) = Ax - b$ with an arbitrary starting value x_0 and an approximate inverse $R \approx A^{-1}$ of A. If there is an index k with $[x]_{k+1} \mathring{\subset} [x]_k$ (the $\mathring{\subset}$ operator denotes that $[x]_{k+1}$ is included in the interior of $[x]_k$), then the matrices R and A are regular, and there is a unique solution x of the system $Ax = b$ with $x \in [x]_{k+1}$. We assume that $Ax = b$ is a dense square linear system and we do not consider any special structure of the elements of A.

3 Tools and Solvers for Dense Linear Systems of Equations

This Section presents Tools and Solvers available to solve dense linear systems of equations. First we will present optimized tools that do not provide verified solutions (see Section 3.1). Section 3.2 shows some verified solver also for dense linear systems of equations. Finally, Section 3.3 discuss the advantages and disadvantages of these tools and solvers.

3.1 Optimized Tools and Solvers

LAPACK is a fortran library for numerical linear algebra. This package includes numerical algorithms for the more common linear algebra problems in scientific computing (solving linear equations, linear least squares, and eigenvalue problems for dense and banded systems).

The numerical algorithms in LAPACK are based on BLAS routines. The BLAS is composed of routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations. BLAS routines are efficient, portable, and widely available. They use block-matrix operations, such as matrix-multiply in inner loops to achieve high performance. These operations improve the performance by increasing the granularity of the computations and keeping the most frequently accessed subregions of a matrix in the fastest level of memory [9].

The SCALAPACK library includes a subset of LAPACK routines redesigned for distributed memory computers. Like LAPACK, the SCALAPACK routines are based on block-partitioned algorithms in order to minimize the frequency of data movement between different levels of the memory hierarchy. (For such machines, the memory hierarchy includes the off-processor memory of other processors, in addition to the hierarchy of registers, cache, and local memory on each processor.)

The fundamental building blocks of the SCALAPACK library use PBLAS, the distributed memory versions of the Level 1, 2 and 3 BLAS, BLACS, a set of Basic Linear Algebra Communication Subprograms for communication tasks that arise frequently in parallel linear algebra computations. In the SCALAPACK routines, all interprocessor communication occurs within the PBLAS and the BLACS.

3.2 Verified Tools and Solvers

There is a multitude of tools and algorithms that provide verified computing. Among them, an option is C-XSC. C-XSC is a free and portable programming environment for C and C++ programming languages, offering high accuracy and automatic verified results. This programming tool allows the solution of many standard problems with reliable results. The MATLAB [32] toolbox for self-verified algorithms, INTLAB, is also an option. Like C-XSC, it also provides interval arithmetic for real and complex data including vectors and matrices, interval arithmetic for real and complex sparse matrices, rigorous real interval standard functions, rigorous complex interval standard functions, rigorous input/output, accurate summation, dot product and matrix-vector residuals, multiple precision interval arithmetic with error bounds, and more. However, INTLAB can be used just together with the commercial MATLAB environment, which can increase the costs to prohibitive values.

3.3 Comparison

Each tool has its advantages and disadvantages, some are more accurate and some are faster. Another important aspect is the availability of these tools. A comparison among C-XSC, INTLAB and LAPACK was performed.

Both C-XSC and LAPACK are free libraries and can be used for educational and scientific purposes. They can be downloaded from internet and used without any restrictions. INTLAB, otherwise, cannot be used freely. INTLAB itself has an open source, but to be able to use INTLAB you have to buy the commercial product MATLAB. This is a big disadvantage which can make the costs of using INTLAB prohibitively large.

C-XSC and INLAB present an interval as result. Both present enclosures of the exact result. In several test cases, C-XSC presented a point interval, while INTLAB presented some uncertainty digits. The verified result makes possible to evaluate how good the floating-point approximation is. LAPACK in the other hand, does not provide a verified result, just an approximation.

As expected, LAPACK presented the best performance. However it gives just an approximation of the correct result, and not an inclusion as INTLAB and C-XSC. INTLAB is based on BLAS, therefore it presents also a good performance comparing with C-XSC. The performance presented by C-XSC is not so optimal because the algorithm uses special variables (data type dotprecision), which are simulated in software to achieve high accuracy.

The results show that C-XSC has the most reliable results and the highest accuracy. LAPACK is the one that presents the best performance, but results are not verified, and in some cases less accurate. INTLAB is the best compromise between performance and accuracy. However, as said before, it requires MATLAB which is not free. The tests show that the method used in C-XSC is a good choice, but it should be optimized to gain performance.

4 Parallel Approach

To implement the parallel version of Algorithm 1, we used an approach for cluster architectures with message passing programming model (MPI [31]) and the highly optimized library PBLAS and SCALAPACK. Clusters of computers are considered a good option to achieve better performance without using parallel programming models oriented to very expensive (but not frequently used) machines. A parallel version for this algorithm runs on distributed processors, requiring communication among the processors connected by a fast network and the communication library.

The self-verified method presented above is divided in several steps. By tests, the computation of R, the approximate inverse of matrix A, takes more than 50% of the total processing time. Similarly, the computation of the interval matrix $[C]$ that contains the exact value of $I - RA$ (iterative refinement) takes more than 40% of the total time, since matrix multiplication requires O(n^3) execution time, and the other operations are mostly vector or matrix-vector operations which require at most O(n^2). Both operations could be implemented using SCALAPACK (R calculation) and PBLAS (C calculation).

A parallel version of the self-verified method for solving linear systems was presented in [20, 21]. In this paper we propose the following improvements aiming at a better performance:

- Calculate R using just floating-point operations;
- Avoid the use of C-XSC elements that could slow down the execution;

- Use of the fast and highly optimized libraries: BLAS and LAPACK in the first sequential version (for the parallel version PBLAS and SCALAPACK respectively);
- Use of both interval arithmetics: infimum-supremum and midpoint-radius (as proposed by Rump [29]);
- Use of techniques to avoid the switching of rounding mode in infimum-supremum operations (proposed by Bohlender [4, 5]).

To find the best arithmetic for this method, the sequential algorithms for point and interval input data were written using both infimum-supremum and midpoint-radius arithmetic. The performance tests showed that the midpoint-radius algorithm needs approximately the same time to solve a linear system with point or interval input data, while the infimum-supremum algorithm needs much more time in the interval case, since the interval multiplication must be implemented with case distinctions and optimized functions from BLAS cannot be used. Therefore, midpoint-radius arithmetic was chosen for the parallel implementation.

The parallel implementation uses the following SCALAPACK/PBLAS routines in Algorithm 1:

- SCALAPACK
  - *pdgetrf*: for the LU decomposition (matrix R on step 1);
  - *pdgetri*: to find the approximation of the inverse matrix (matrix R on step 1).
- PBLAS
  - *pdgemm*: matrix-matrix multiplication (matrix C on step 4);
  - *pdgemv*: matrix-vector multiplication (many steps: 2, 3 and 8 to find the vectors x, z and w);.

It is important to remember that behind every midpoint-radius interval operation more than one floating-point operation should be done using the appropriate rounding mode. The matrix multiplication $R * A$ from step 4 needs the following operations:

Algorithm 2. Midpoint-radius matrix multiplication in $\mathbb{F}^n$

1: $\tilde{c_1} = \bigtriangledown(R \cdot mid(A))$
2: $\tilde{c_2} = \bigtriangleup(R \cdot mid(A))$
3: $\tilde{c} = \bigtriangleup(\tilde{c_1} + 0.5(\tilde{c_2} - \tilde{c_1}))$
4: $\tilde{\gamma} = \bigtriangleup(\tilde{c} - \tilde{c_1}) + |R| \cdot rad(A)$

In this case, the routine PDGEMM of PBLAS would be called three times, one for step 1, one for step 2, and one for step 4.

5 Results

This section presents some experimental results of the new parallel implementation. Three different tests were executed. Performance tests are presented in Section 5.1, accuracy tests are shown in Section 5.2 and finally a real problem test is discussed in Section 5.3. This set of experiments represents a wide range of aspects that should be tested in parallel verified algorithms.

5.1 Performance

Performance analysis of this parallel solver was carried out varying the order of input matrix A. Matrices with three different orders were used as test cases: 2.500×2.500, 5.000×5.000, and 10.000×10.000. For each of those matrices, executions with the number of processors varying from 1 to 64 were performed. All matrices are dense and were specifically generated for these experiments and are composed of random numbers. This random numbers were generated with the function *rand()*.

The results presented in this section were obtained on the HP XC6000, the high performance computer of the federal state Baden-Württemberg. The HP XC6000 is a distributed memory parallel computer with 128 nodes all in all; 108 nodes consist of 2 Intel Itanium2 processors with a frequency of 1.5 GHz and 12 nodes consist of 8 Intel Itanium2 processors with a frequency of 1.6 GHz. All nodes own local memory, local disks and network adapters. Thus the theoretical peak performance of the system is 1.9 TFLOPS. The main memory above all compute nodes is about 2 TB. All nodes are connected to the Quadrics QsNet II interconnect that shows a high bandwidth of more than 800 MB/s and a low latency. The basic operating system on each node is HP XC Linux for High Performance Computing (HPC), the compiler used was the gcc and the MKL version 10.0.011 was used for an optimized version of libraries SCALAPACK and PBLAS.

Figure 1 presents a comparison of the speed-ups achieved for the tested matrices. The proposed parallel solution presents a good scalability and improves the performance of the application. As expected, the larger the input matrix, the better is the speed-up achieved.

For larger dimensions, the speed-up is almost linear. In some cases, like for dimension 10.000 and 16 processors, we found a super linear speed-up. This is possible due to cache effects resulting from the different memory hierarchies of a modern computer. In this case, the size of accumulated caches from different processors can also change, and sometimes all data can fit into caches and the memory access time reduces dramatically, leading to a drastic speed-up. It is understandable that this effect occurs in

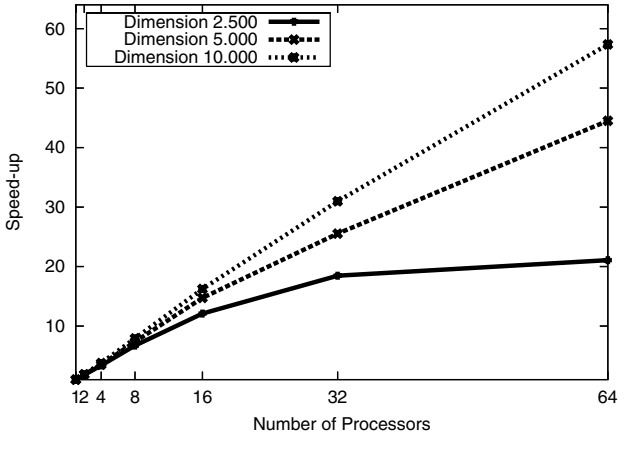


Fig. 1. Experimental results - Speed-up

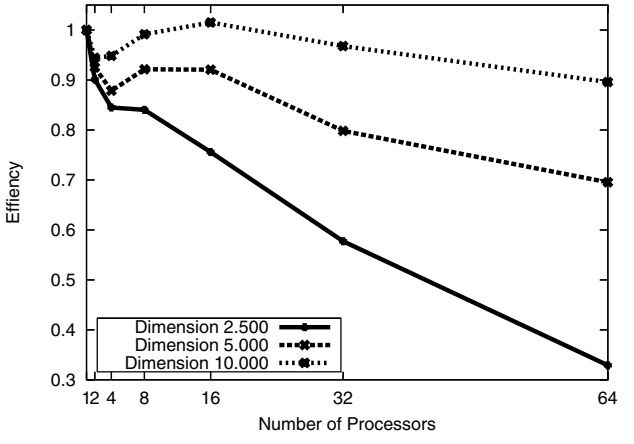


Fig. 2. Experimental results - Efficiency

this implementation since PBLAS and SCALAPACK were optimized to make the best possible use of cache hierarchies.

Results for more than 64 processors were not presented for these test cases since beyond this threshold the performance started to drop down. As can be seen in figure 2, for 64 processors, the efficiency drops significatively, for the test case with dimension 2.500, it drops under 35%.

5.2 Accuracy

The accuracy depends on the condition number of the matrix A. For well conditioned problems, the new algorithm may deliver a very accurate result with up to 16 correct digits.

For example supposing A is a 10x10 point matrix with condition number $4.06 \cdot 10^{+02}$ and b is a vector where each position has the value 1.0. The solution for x delivered by the developed algorithm is presented in Table 1. Analyzing this results, it is possible to see that the radius of the solution is very small, and therefore it is a very accurate solution.

The new implementation also finds an inclusion for ill-conditioned dense matrices, but the accuracy may vary depending on the condition number as presented in Table 2.

It is important to mention that this relation between condition number and diameter of the resulting interval was found for a special class of matrix: square, dense with random numbers.

A well-known example of ill-conditioned matrix are the Boothroyd/Dekker matrices that are defined by the following formula:

$$A_{ij} = \binom{\text{n+i-1}}{\text{i-1}} \times \binom{\text{n-1}}{\text{n-j}} \times \tfrac{n}{i+j-1}, b_i = i, \forall i, j = 1..n,$$

For $n = 10$ this matrix has a condition number of $1.09 \cdot 10^{+15}$. The result found by this parallel solver is presented in Table 3.

Table 1. Midpoint-radius and the equivalent infimum-supremum result

Res	Midpoint	Radius
x[0] =	-1.81645396742261731e-05	3.30189340310308432e-18
x[1] =	-1.81103764097278691e-06	1.27042988556363316e-18
x[2] =	-3.50284396008609789e-06	8.65422361039543574e-19
x[...] =	...	...
x[8] =	-2.65213537238245186e-06	1.12671706774209183e-18
x[9] =	3.01161008622528871e-05	4.81485187791373545e-18

Res	Infimum	Supremum
x[0] =	-0.00001816453967423	-0.00001816453967422
x[1] =	-0.00000181103764097	-0.00000181103764097
x[2] =	-0.00000350284396009	-0.00000350284396009
x[...] =	...	...
x[8] =	-0.00000265213537238	-0.00000265213537238
x[9] =	0.00003011610086225	0.00003011610086226

Table 2. Relation between condition number and diameter

Condition number	diameter
10^1	10^{-14}
10^2	10^{-13}
10^3	10^{-12}
10^4	10^{-11}
10^5	10^{-10}
10^6	10^{-9}
10^7	10^{-8}
10^8	10^{-7}
10^9	10^{-6}

Table 3. Results for the Boothroyd/Dekker 10x10 matrix

Res	Midpoint	Radius	Infimum	Supremum
x[0]	5.4711703e-07	2.8569585e-06	-0.0000023	0.0000034
x[1]	9.9999473e-01	2.7454332e-05	0.9999672	1.0000221
x[2]	-1.9999718e+00	1.4633209e-04	-2.0001182	-1.9998255
x[3]	2.9998902e+00	5.7081064e-04	2.9993194	3.0004611
x[4]	-3.9996506e+00	1.8174619e-03	-4.0014680	-3.9978331
x[5]	4.9990383e+00	5.0008707e-03	4.9940374	5.0040392
x[6]	-5.9976307e+00	1.2322380e-02	-6.0099531	-5.9853083
x[7]	6.9946526e+00	2.7799205e-02	6.9668534	7.0224518
x[8]	-7.9887617e+00	5.8434700e-02	-8.0471964	-7.9303270
x[9]	8.9777416e+00	1.1572349e-01	8.8620181	9.0934651

Matrix QH1484: Quebec Hydroelectric Power System

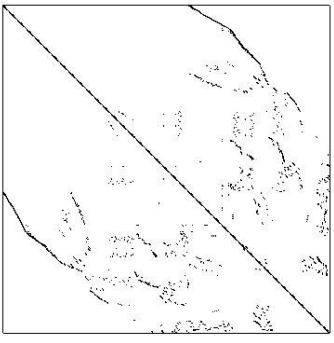


Fig. 3. Structure view of matrix QH1484

Table 4. Results for QH1484: Quebec Hydroelectric Power System

Res	Midpoint	Radius	Infimum	Supremum
x[0]	$-4.1415310 \cdot 10^{+00}$	$2.0242898 \cdot 10^{-07}$	$-4.1415312 \cdot 10^{+00}$	$-4.1415308 \cdot 10^{+00}$
x[1]	$-2.1936961 \cdot 10^{+00}$	$2.3526014 \cdot 10^{-07}$	$-2.1936964 \cdot 10^{+00}$	$-2.1936959 \cdot 10^{+00}$
x[2]	$-4.1417322 \cdot 10^{+00}$	$2.0242898 \cdot 10^{-07}$	$-4.1417324 \cdot 10^{+00}$	$-4.1417320 \cdot 10^{+00}$
x[3]	$-2.1954030 \cdot 10^{+00}$	$2.3526014 \cdot 10^{-07}$	$-2.1954032 \cdot 10^{+00}$	$-2.1954028 \cdot 10^{+00}$
x[...]	...	...	...	...

As expected for an ill-conditioned problem, the accuracy of the results is not the same as for a well-conditioned problem. It is important to remark that even if the result has an average diameter of $4.436911 \cdot 10^{-02}$, it is an inclusion. In other words, it is a verified result.

5.3 Real Problem

For a real problem test, the used matrix is from the application of the Hydro-Quebec power systems' small-signal model, used for power systems simulations [2]. This problem uses a square 1484 x 1484 real unsymmetric matrix, with 6110 entries (1126 diagonals, 2492 below diagonal, 2492 above diagonal) as can be seen in Figure 3.

The presented solver was written for dense systems, therefore, this sparse systems will be treated as a dense system. No special method or data storage was used/done concerning the sparsity of this systems.

The first elements of the result vector found for this problem with conditional number $5.57 \cdot 10^{17}$ is presented in Table 4.

Despite it is an ill-conditioned problem, the average diameter of the interval results found by this solver was $1.26 \cdot 10^{-8}$. This is a very accurate result for such an ill-conditioned problem.

Table 5. Real problem Execution time and Speed-up

Number of proc.	Execution Time (sec)	Speed-up
1	8.728	1.000
2	4.307	2.026
4	2.186	3.992
8	1.419	6.150

As shown in Section 5.1 for dimension 2.500 the efficiency drops very fast for small matrices. Therefore we measure the execution time of this matrix until 8 processors. The execution time and speed-up for solving this systems of linear equations is presented in Table 5. This performance can be considered very fast to find a verified solution of a linear system.

6 Results and Conclusions

New sequential algorithms based on a verified method using just libraries like BLAS and LAPACK were implemented to achieve better performance. The idea of reducing the switching of rounding mode presented by Bohlender was implemented as well as an optimization of the residuum based on the INTLAB method. In other words, the new implementations join the best aspects of each library.

To ensure that an enclosure will be found, interval arithmetic was used. To find the best arithmetic for this method, the sequential algorithms for point input data were written using both infimum-supremum and midpoint-radius arithmetic. The performance tests showed that the midpoint-radius algorithm needs approximately the same time to solve a linear system with point and interval input data, while the infimum-supremum algorithm needs much more time in the interval case, since the interval multiplication must be new implemented and in this case optimized functions from BLAS cannot be used. Therefore, midpoint-radius arithmetic was chosen for the parallel implementation.

Aiming at a better performance, the algorithm was parallelized using the libraries SCALAPACK and PBLAS. The performance results showed that the parallel implementation leads to nearly perfect speed-up in a wide range of processor numbers for large dimensions. This is a very significant result for clusters of computers.

One important advantage of the presented algorithm is the ability to find a solution even for ill-conditioned problems while most algorithms may lead to an incorrect result when it is too ill-conditioned (above condition number 10^8). The accuracy of the results in many cases depends on the condition number. However, the result of this method is always an inclusion, given the guarantee that the correct result is inside the interval.

Our main contribution is to provide a free, fast, reliable and accurate solver for dense linear systems and to increase the use of verified computing through its optimization and parallelization, once without parallel techniques it becomes the bottleneck of an application.

Among the ideas for future work the use of interval input data is the first to be implemented. The parallelization of a verified method for solving sparse matrices is also a goal for the future. Since many real problems are modeled using these systems,

it is a natural choice to implement such methods, joining the benefits of verified and parallel computing.

References

1. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.: LAPACK Users' Guide. In: Society for Industrial and Applied Mathematics, 3rd edn., Philadelphia, PA (1999)
2. Bai, A., Day, D., Dongarra, J.: A test matrix collection for non-hermitian eigenvalue problems. Technical report, Knoxville, TN, USA (1997)
3. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK Users' Guide. In: Society for Industrial and Applied Mathematics, Philadelphia, PA (1997)
4. Bohlender, G.: Faster Interval Computations Through Minimized Switching of Rounding Modes. In: PUCRS, Porto Alegre, Brazil (2002)
5. Bohlender, G., Kolberg, M., Claudio, D.: Modifications to Expression Evaluation in C-XSC. Technical report, preprint BUW-WRSWT 2005/5, Universität Wuppertal, DE, SCAN 2004, Fukuoka, Japan (2005)
6. Claudio, D.M., Marins, J.M.: Cálculo Numérico Computacional: Teoria e Prática, Editora Atlas S. A., São Paulo (2000)
7. Dongarra, J.J., Duff, I.S., Sorensen, D.C., van der Vorst, H.A.: Numerical Linear Algebra for High-performance Computers. SIAM Press, Philadelphia (1998)
8. Dongarra, J.J., Du Croz, J., Duff, I.S., Hammarling, S.: A set of Level 3 Basic Linear Algebra Subprograms. ACM trans. Math. Soft. 16, 1–17 (1990)
9. Dongarra, J.J., Pozo, R., Walker, D.W.: LAPACK++: A design overview of object-oriented extensions for high performance linear algebra. In: Supercomputing 1993. Proceedings, pp. 162–171 (1993)
10. Duff, I.S., van der Vorst, H.A.: Developments and Trends in the Parallel Solution of Linear Systems. Technical Report RAL TR-1999-027, CERFACS, Toulouse, France (1999)
11. Facius, A.: Iterative solution of linear systems with improved arithmetic and result verification. PhD thesis, University of Karlsruhe, Germany (2000)
12. Frommer, A.: Lösung linearer Gleichungssysteme auf Parallelrechnern. Vieweg-Verlag, Universität Karlsruhe, Germany (1990)
13. Gonzalez, P., Cabaleiro, J.C., Pena, T.F.: Solving Sparse Triangular Systems on Distributed Memory Multicomputers. In: Proceedings of the Sixth Euromicro Workshop on Parallel and Distributed Processing, pp. 470–478. IEEE Press, Los Alamitos (1998)
14. Hammer, R., Ratz, D., Kulisch, U., Hocks, M.: C++ Toolbox for Verified Scientific Computing I: Basic Numerical Problems. Springer, New York (1997)
15. Hölbig, C.A., Morandi Júnior, P.S., Alcalde, B.F.K., Diverio, T.A.: Selfverifying Solvers for Linear Systems of Equations in C-XSC. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2004. LNCS, vol. 3019, pp. 292–297. Springer, Heidelberg (2004)
16. Hölbig, C.A., Krämer, W., Diverio, T.A.: An Accurate and Efficient Selfverifying Solver for Systems with Banded Coefficient Matrix. In: Proceedings of Parallel Computing (PARCO), Germany, pp. 283–290 (September 2003)
17. INTLAB. INTerval LABoratory, `http://www.ti3.tu-harburg.de/~rump/intlab/`
18. Kersten, T.: Verifizierende rechnerinvariante Numerikmodule. PhD thesis, University of Karlsruhe, Germany (1998)

19. Klatte, R., Kulisch, U., Lawo, C., Rauch, R., Wiethoff, A.: C-XSC- A C++ Class Library for Extended Scientific Computing. Springer, Berlin (1993)
20. Kolberg, M., Baldo, L., Velho, P., Fernandes, L.G., Claudio, D.: Optimizing a Parallel Self-verified Method for Solving Linear Systems. In: Kågström, B., Elmroth, E., Dongarra, J., Waśniewski, J. (eds.) PARA 2006. LNCS, vol. 4699, pp. 949–955. Springer, Heidelberg (2007)
21. Kolberg, M., Baldo, L., Velho, P., Webber, T., Fernandes, L.G., Fernandes, P., Claudio, D.: Parallel Selfverified Method for Solving Linear Systems. In: Daydé, M., Palma, J.M.L.M., Coutinho, Á.L.G.A., Pacitti, E., Lopes, J.C. (eds.) VECPAR 2006. LNCS, vol. 4395, pp. 179–190. Springer, Heidelberg (2007)
22. Kulisch, U., Miranker, W.L.: Computer Arithmetic in Theory and Practice. Academic Press, New York (1981)
23. LAPACK. Linear Algebra Package, http://www.cs.colorado.edu/~jessup/lapack/
24. Lo, G.C., Saad, Y.: Iterative Solution of General Sparse Linear Systems on Clusters of Workstations. Technical Report umsi-96-117, msi, uofmad (1996)
25. Ogita, T., Rump, S.M., Oishi, S.: Accurate Sum and Dot Product with Applications. In: 2004 IEEE International Symposium on Computer Aided Control Systems Design, Taipei, Taiwan, September 2004, pp. 152–155. IEEE Press, Los Alamitos (2004)
26. Ogita, T., Rump, S.M., Oishi, S.: Accurate Sum and Dot Product. SIAM Journal on Scientific Computing 26(6), 1955–1988 (2005)
27. Pan, V., Reif, J.: Fast and Efficient Parallel Solution of Dense Linear Systems. Computers & Mathematics with Applications 17(11), 1481–1491 (1989)
28. Rump, S.: INTLAB - INterval LABoratory. Developments in Reliable Computing, pp. 77–104 (1998), http://www.ti3.tu-harburg.de/~rump/intlab
29. Rump, S.: Fast and Parallel Interval Arithmetic. Bit Numerical Mathematics 39(3), 534–554 (1999)
30. Rump, S.M.: Solving Algebraic Problems with High Accuracy. In: IMACS World Congress, pp. 299–300 (1982)
31. Snir, M., Otto, S., Huss-Lederman, S., Walker, D.W., Dongarra, J.: MPI: The Complete Reference. MIT Press, Cambridge (1996)
32. Inc. (publisher) The MathWorks. MATLAB, The Language of Technical Computing (2001), http://www.mathworks.com
33. Wiethoff, A.: Verifizierte globale Optimierung auf Parallelrechnern. PhD thesis, University of Karlsruhe, Germany (1997)
34. Zhang, J., Maple, C.: Parallel Solutions of Large Dense Linear Systems Using MPI. In: International Conference on Parallel Computing in Electrical Engineering, PARELEC 2002, pp. 312–317. IEEE Computer Society Press, Los Alamitos (2002)

Parallelisation of the CFD Code of a CFD-NWP Coupled System for the Simulation of Atmospheric Flows over Complex Terrain

F.A. Castro[1,2], C.M.P. Silva Santos[1], and J.M.L.M. Palma[1]

[1] CEsA - Centre for Wind Energy and Atmospheric Flows
FEUP - Faculdade de Engenharia da Universidade do Porto
{csantos,jpalma}@fe.up.pt
[2] ISEP - Instituto Superior de Engenharia do Porto
fac@isep.ipp.pt

Abstract. The sequential simulation of atmospheric flows over complex terrain using Computational Fluid Dynamics tools (CFD) leads normally to very large time-consuming runs. With the present day processors only the power available using parallel computers is enough to produce a true prediction using CFD tools, i.e. running the code faster than the evolution of the real weather. In the present work, the parallelisation strategy used to produce the parallel version of the VENTOS® CFD code is shown. A sample of the results included in the present abstract is enough to show the code behaviour as a function of the number of sub-domains, both number and direction along which the domain splitting occurs, and their implications on both the iteration number and code parallel efficiency.

Keywords: Atmospheric flows, computational fluid dynamics, domain decomposition, micro and mesoscale coupling, wind power prediction.

1 Introduction

The sequential simulation of atmospheric flows over complex terrain using Computational Fluid Dynamics tools (CFD) leads normally to very large time-consuming runs, when temporal and spatial descriptions of the flows are needed. These are for example the requirements of the simulations to be used in the Short Term Prediction of the atmospheric flows over complex terrain. The Short Term Prediction means predictions of time periods of 1–3 days, typically, and, in the present context, requires the use of an operational Numerical Weather Prediction (NWP) program coupled to a CFD code for performing a zooming effect over the NWP results, which will produce results with higher accuracy. With the present day processors only the power available using parallel computers is enough to produce a true prediction using CFD tools, i.e. running the code faster than the evolution of the real weather.

In the present work, the parallelisation strategy used to produce the parallel version of the VENTOS® CFD code [1,2] is presented. This code has been used

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 27–38, 2008.

with success in the site assessment of wind farms and so the natural choice for us to couple with mesoscale codes to produce a Short Term Prediction tool.

In the following sections, we show the fundamental equations being solved (section 2). In section 3 the parallelisation strategy is presented and in section 4 the results are discussed. Conclusions are presented in section 5

2 Mathematical Model

This section covers the fundamental equations, coordinate transformation and the numerical techniques used in the current study. A more complete description of the model can be found in [2].

The continuity (1), the momentum (2), the potential temperature transport (3) and the turbulence model equations (4 and 5) were written in tensor notation for a generic coordinate system.

$$\frac{\partial\left(\rho U_j \beta_k^j\right)}{\partial \xi^j} = 0, \tag{1}$$

$$\frac{\partial\left(\rho J U_i\right)}{\partial t} + \frac{\partial}{\partial \xi^j}\left(\rho U_k U_i \beta_k^j\right) = -\frac{\partial}{\partial \xi^j}\left(P \beta_i^j\right) + \frac{\partial}{\partial \xi^j}\left[\left(\tau_{ki} + \sigma_{ki}\right)\beta_k^j\right], \tag{2}$$

$$\frac{\partial\left(\rho c_p J \theta\right)}{\partial t} + \frac{\partial}{\partial \xi^j}\left(\rho c_p U_k \theta \beta_k^j\right) = \frac{\partial}{\partial \xi^j}\left[\frac{K_\theta}{J}\frac{\partial \theta}{\partial \xi^m}\beta_k^m \beta_k^j\right], \tag{3}$$

where

$$\tau_{ij} = \frac{\mu}{J}\left(\frac{\partial U_i}{\partial \xi^m}\beta_j^m + \frac{\partial U_j}{\partial \xi^m}\beta_i^m\right)$$

and

$$\sigma_{ij} = -\frac{2}{3}\rho k \delta_{ij} + \frac{\mu_t}{J}\left(\frac{\partial U_i}{\partial \xi^m}\beta_j^m + \frac{\partial U_j}{\partial \xi^m}\beta_i^m\right).$$

The eddy viscosity was given by $\mu_t = \rho C_\mu k^2/\epsilon$, where k and ϵ were obtained from

$$\frac{\partial\left(\rho J k\right)}{\partial t} + \frac{\partial}{\partial \xi^j}\left(\rho U_k k \beta_k^j\right) = \frac{\partial}{\partial \xi^j}\left[\frac{1}{J}\left(\mu + \frac{\mu_t}{\sigma_k}\right)\frac{\partial k}{\partial \xi^m}\beta_k^m \beta_k^j\right] + J\left(P_k - \rho\epsilon\right) \tag{4}$$

and

$$\frac{\partial\left(\rho J \epsilon\right)}{\partial t} + \frac{\partial}{\partial \xi^j}\left(\rho U_k \epsilon \beta_k^j\right) = \frac{\partial}{\partial \xi^j}\left[\frac{1}{J}\left(\mu + \frac{\mu_t}{\sigma_\epsilon}\right)\frac{\partial \epsilon}{\partial \xi^m}\beta_k^m \beta_k^j\right] + J\left(\frac{C_1 \epsilon}{k}P_k - \frac{C_2 \rho \epsilon^2}{k}\right), \tag{5}$$

where

$$P_k = \sigma_{ij}\frac{1}{J}\frac{\partial U_i}{\partial \xi^m}\beta_j^m. \tag{6}$$

$$\underline{C_1 = 1.44\ C_2 = 1.92\ C_\mu = 0.033\ \sigma_k = 1.0\ \sigma_\epsilon = 1.85}$$

In the above equations, U, P and θ are the Reynolds averaged velocities, pressure and dry potential temperature, ρ and μ are the air density and molecular viscosity, and k and ϵ are the turbulence kinetic energy and its dissipation rate, whose transport equations (4 and 5) and constants C_1, C_2 and σ_k were set as in [3], whereas the constants C_μ and σ_ϵ followed the recommendations by [4]. The turbulent Prandtl number was chosen equal to unity which implies $K_\theta = \mu_t c_p$.

The coordinate system is defined by transforming a physical Cartesian coordinate system x^i into a computational system ξ^i, where the co-factors are $\beta_k^j = J\partial\xi^j/\partial x^k$ and J is the determinant of the Jacobian matrix of the coordinate transformation (cf., [5]). This transformation makes it relatively simple to treat the boundary conditions and to use a structured mesh, where the physical domain boundaries are the coordinate surfaces following the topography.

The transport equations (1 – 5) were discretized by finite volume techniques using a central differencing scheme for the diffusive terms. The advective terms were discretized by the hybrid scheme. In the case of the momentum equations (2), an alternative 3rd order truncation scheme was used for the advective terms, identical to the QUICK scheme for non-uniform meshes [6].

The resulting set of coupled algebraic equations was solved using the SIMPLE algorithm of [6] and the Tri-Diagonal Matrix Algorithm solver, to take advantage of the structure of the coefficient matrices. The pressure/velocity coupling in non-staggered meshes was treated following the pressure-weighted interpolation as in [7] and [8].

The equations can be solved in time-dependent mode, using a second order implicit scheme, or in steady state formulation, in which case the time derivatives in equations (2) – (5) were dropped. In either case the equations were solved until mass and momentum could be satisfied to a dimensionless residual below $5{\times}10^{-4}$.

The development and validation of the VENTOS® code for a series of atmospheric flows, as well as details on boundary and initial conditions, can be found in [1] and [2].

3 Parallelisation Technique

The algorithm of the CFD code used in this work can be said to contain two iterative levels: an inner iteration where the solution of each equation is iterated a small number of sweeps – to each equation corresponds a different subroutine; and an outer iteration which contains all the inner levels and is responsible for the coupling between equations. The equations for different variables are solved in a segregated manner, one after the other. In the inner iterations, we use the Tri-Diagonal Matrix Algorithm (TDMA) (see, for example, [9]) to solve the various algebraic systems. The Message Passing Interface library (MPI) is used to implement the communication between processes.

The code was parallelised using a domain decomposition strategy, where the physical domain, discretised by a mesh of control volumes with a central node, was decomposed into several sub-domains, each being calculated in a dedicated

Table 1. Speed-up and efficiency results for several partition schemes. Two sets are shown, averaged results until convergence and averaged results at the end of 1000 iterations. Time is displayed in seconds and efficiency in %. Speedup corresponds to the ratio of execution times and NP represents the number of processors.

		Until convergence				Per 1000 iterations		
Partitions	NP	Iterations	Time	Efficiency	Speed-Up	Time	Efficiency	Speed-Up
1×1×1	1	1060	18504	-	-	17456	-	-
2×1×1	2	1129	11666	79.3	1.6	10333	84.5	1.7
1×2×1	2	984	9571	96.7	1.9	9727	89.7	1.8
1×1×2	2	1160	9138	101.3	2.0	7877	110.8	2.2
3×1×1	3	1404	16936	36.4	1.1	12063	48.2	1.5
1×3×1	3	968	7511	82.1	2.5	7759	75.0	2.3
1×1×3	3	1181	7779	79.3	2.4	6587	88.3	2.7
4×1×1	4	1815	15425	30.0	1.2	8499	51.4	2.1
2×2×1	4	1068	6997	66.1	2.6	6552	66.6	2.7
2×1×2	4	1194	5184	89.2	3.6	4341	100.5	4.0
1×4×1	4	979	4827	95.8	3.8	4931	88.5	3.5
1×2×2	4	1115	4834	95.7	3.8	4335	100.7	4.0
5×1×1	5	2302	11004	33.6	1.7	4780	73.0	3.7
1×5×1	5	1087	3108	119.1	5.9	2860	122.1	6.1
6×1×1	6	3620	11610	26.6	1.6	3207	90.7	5.4
3×2×1	6	1293	6120	50.4	3.0	4733	61.5	3.7
3×1×2	6	1271	6878	44.8	2.7	5411	53.8	3.2
2×3×1	6	1085	3860	79.9	4.8	3558	81.8	4.9
1×3×2	6	1055	2507	123.0	7.4	2376	122.4	7.4
4×2×1	8	1677	3741	61.8	4.9	2285	95.5	7.6
2×4×1	8	1114	2815	82.2	6.6	2527	86.3	6.9
2×2×2	8	1148	3500	66.1	5.3	3049	71.6	5.7
3×3×1	9	1316	2484	82.8	7.4	1888	102.8	9.3
5×2×1	10	2089	4989	37.1	3.7	2447	71.4	7.1
2×3×2	12	1103	1932	79.8	9.6	1752	83.1	10.0
5×3×1	15	2116	2863	43.1	6.5	1353	86.0	12.9
3×3×2	18	1181	1381	74.5	13.4	1169	83.0	14.9
2×5×2	20	1101	1094	84.6	16.9	994	87.8	17.6

processor. Inside each sub-domain, the code works essentially as its sequential version plus the necessary communications to exchange the boundary information with neighbouring sub-domains. This physical domain decomposition was performed with a fixed overlap of two grid nodes (see figure 1).

Communication between adjoining sub-domains takes place after each sweep of the local TDMA solvers, providing the exchange of the overlapped grid node values. Other communication instances occur at the beginning and end of each of the subroutines, where communication of shared grid node values, momentum fluxes at the control volumes boundaries and some algebraic equation coefficients are

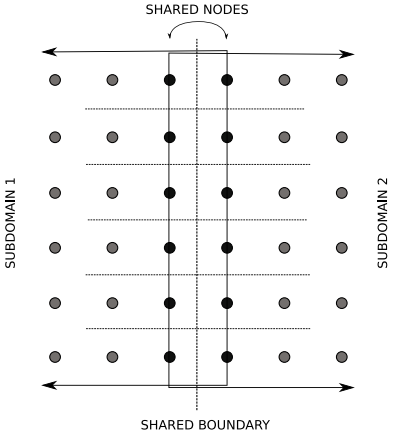


Fig. 1. Representation of overlapped domains

exchanged. For convergence checking, global residuals are constructed by collecting local information from the individual sub-domains, which occurs at the end of the outer iteration loop. Similar techniques can be found, for example, in [10].

All computation is thus parallelised: i.e. not only the algebraic solver, but also the routines that construct the coefficient matrices, the reading of external files and the writing of output files, with the exception of minimal output to standard output to monitor the progress of the simulation and the construction of global residuals and fluxes, which are handled by the master process.

4 Results

4.1 Speed-Up and Efficiency

In this section, we investigate the effect of different partitioning schemes on convergence and computational time, as well as the effect of the size of the mesh on the efficiency of the parallelisation. Speed-up and efficiency results were measured from simulations of a real flow of 60° winds (WNW direction) over a future wind farm at Mendoiro/Bustavade located in the North of Portugal. All simulations were carried out in a cluster with 64 nodes using Intel(R) Xeon(TM) CPU 3.00GHz, and all results shown in this section were obtained by averaging over three different simulations performed at different instances, with the aim of removing oscilations in the performance of the cluster.

The results for several runs, using a mesh of $113 \times 77 \times 45$ ($= 391\,545$) grid nodes and different partition schemes, are presented in table 1 and figures 2–3. In table 1, the number of sub-domains used in each of the computational directions is described in the partitions label, the first column. The total number of processors used in the runs, equal to the number of sub-domains, are shown

in variable NP. Two sets of results are shown: (i) results obtained for converged simulations; and (ii) results after a fixed number of iterations (1000). Whilst the former is included to indicate the speed-up of real applications, the latter reveals the actual speed-up of the code in terms of parallel efficiency (i.e. how much CPU time is being consumed in communication overheads, etc.). With respect to the results using this mesh, one may observe that the computational efficiency per 1000 iterations is quite high and that it does not degrade with an increasing number of processors, (see blue dashed line in figure 2, which is a linear regression curve with a slope of 0.849).

However, when CPU times are obtained from completed simulations, the efficiency decreases due to the larger number of iterations that some partition schemes require. Partitioning often reduces the convergence rate because of the slight decoupling that is introduced by the domain splitting. From table 1, one can see that, in this case, partitioning in the first direction (i.e. schemes $2\times1\times1$, $3\times1\times1$, etc.), which is longitudinal with respect to the flow, has the effect of increasing the number of iterations until convergence. This is not always the case, and is highly dependent on the specific flow features. For other flows, it is slicing the domain horizontally that has a strong impact on convergence, because quantities vary more quickly in the vertical direction near the ground. What should be retained, however, is that, despite some reduction in the convergence rate, the parallelisation efficiency until convergence is still very significant: the blue dashed line in figure 3, has a slope of 0.702.

A small number of partition schemes produced parallelisation efficiencies slightly in excess of the maximum theoretical value of SPEED-UP = NP, which can be confirmed by inspection of table 1 or figure 2. Since this occurred mainly for the finer mesh and for partition schemes with relatively few subdomains (namely, NP $= 2, 4, 5, 6, 9$, which means the size of the subdomains is still fairly large), it is thought that it is not related to issues of memory management. Instead, the explanation is likely to be that the sequential runs were performed in worse computational conditions than these parallel runs (i.e. the computer cluster was heavily loaded, there were filesystem delay issues, etc.), which can lead to some exaggeration of the speed-up of these cases, despite the efforts to minimise this by performing three runs per case.

Figures 2–3 also contain results for a mesh with half the nodes in each direction, $57\times38\times23$ ($= 49\,818$) grid nodes. These data are not tabulated here. It can be seen that for such a smaller case, the parallel efficiency decreases considerably. This is to be expected since the bulk of the computational work performed by the algebraic solver, where most gains are obtained when parallelising, has a much smaller weight in the overall CPU time, when the mesh is small. This is especially true when the number of processes is larger than 6, which means the larger subdomain has less than 8000 grid nodes.

The increase in calculation speed obtained by the present parallelisation strategy was sufficient to produce fast enough calculations, when using for example 9 processors, that enabled us to forecast in 2.5 days a forecast horizon of 5 days.

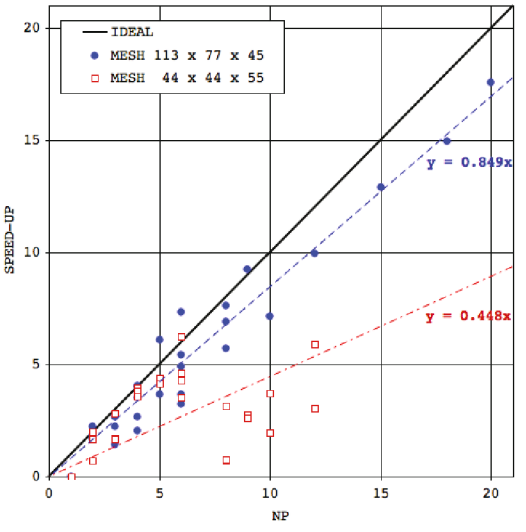


Fig. 2. Speed-up versus number of processors, NP, per 1000 iterations for two mesh sizes

4.2 Wind Prediction Results

In this section, results of the coupling between the parallel version of VENTOS and the mesoscale code are presented and compared to field measurements. This approach consists of two different simulations, mesoscale and microscale (CFD), with one way coupling linking them. The objective is that the mesoscale model, feed but planetary weather simulations, provides an accurate numerical weather prediction for a large (130×130 km) area, encompassing the wind farm. The CFD code then uses the mesoscale results as boundary conditions, bringing additional accuracy due to the higher resolution meshes and more accurate terrain representation.

A test case for this procedure was performed for the flow over the Mendoiro/Bustavade wind farm site. Three different occasions were selected from the year 2006; two winter occasions, 10 to 15 of January and 1 to 5 of December and a Spring/Summer occasion, 1 to 5 of June. The 2006 year was chosen only because of the availability of both experimental and NCEP (National Centers for Environmental Prediction) results, the latter used to drive the WRF simulations (Weather Research Forecast code from the National Center for Atmospheric Research (NCAR) and others, see `http://www.wrf-model.org`).

The VENTOS simulations used a mesh of $39 \times 39 \times 55$ grid nodes with $3 \times 3 \times 1$ partitions, covering a domain of 22×22 km in the horizontal and 7500 m in the vertical. The mesh was almost uniform in the horizontal directions and concentrated near the ground in the vertical direction, where control volumes of 5 m height were used. The mesoscale simulations, using WRF-ARW core (version 2.2.1 and WPS pre-processing system), were done using a mesh of $44 \times 44 \times 31$ grid nodes in a single processor, using a uniform mesh spacing in the

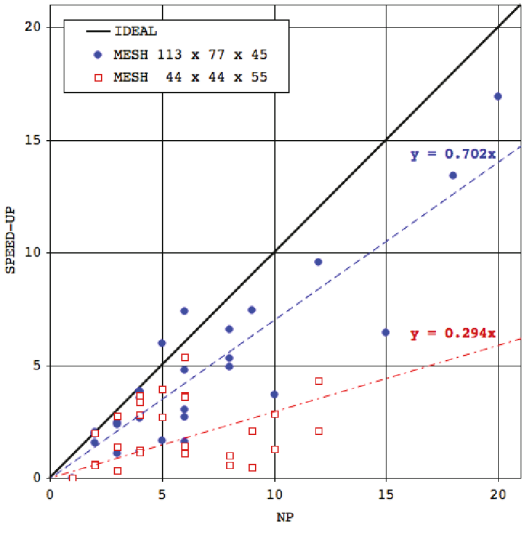


Fig. 3. Speed-up versus number of processors, NP, until convergence for two mesh sizes

horizontal directions of 3000 m and 31 eta levels in the vertical, reaching almost 20 km in height. The time steps of VENTOS and WRF were 2 s and 10 s respectively.

Representations of the horizontal extensions and meshes used in both codes are shown in figures 4 and 5. The zooming effect produced by VENTOS is near 6×, as can be seen in figure 4. From figure 5, it can be seen that VENTOS uses the WRF topography at the boundaries, being then operated a transition to a higher resolution description of the topography. This transition occurs inside a region surrounding the VENTOS domain of 3000 m.

To produce the VENTOS results the WRF simulations were first performed. In the WRF simulations, the results were written to disk every 20 minutes, i.e. every 120 time steps. These results were then interpolated to the boundaries of the VENTOS mesh, producing files corresponding to each of the 20 minutes snapshots from WRF. During the VENTOS simulations the boundary conditions were updated every time step using linear interpolations in time between WRF snapshots. All the simulations (VENTOS and WRF) were performed in the aforementioned cluster during normal operation days. The VENTOS simulations took nearly 1 cluster day for every 5 days of real time when the $3 \times 3 \times 1$ partitioning was used, whilst the sequential version would require more cluster days than days of real time.

For the site under study results from three meteorological masts operated with cup anemometers at 60 m above the ground level were available. For this study we present only results for one mast, known as PORT267.

The VENTOS and WRF results for the horizontal velocity magnitude (V_h) are compared with cup anemometer results in figures 6-8. Figure 9 compares the VENTOS and WRF predictions for the potential temperature (θ).

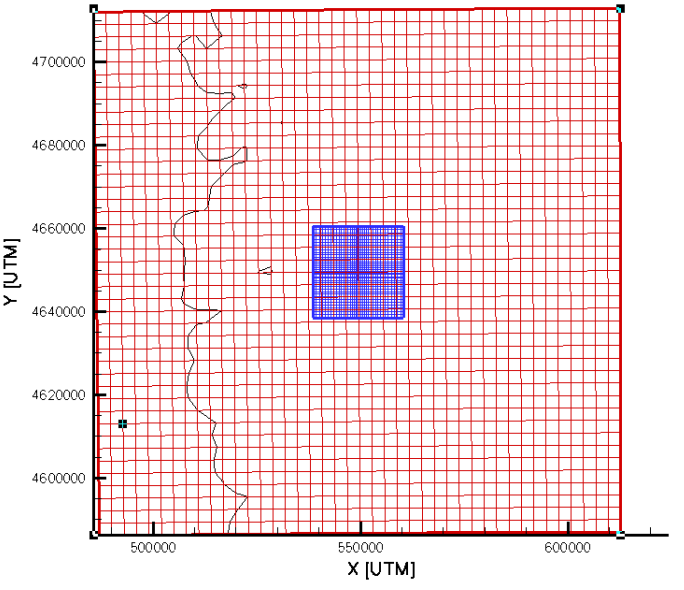


Fig. 4. Meshes used by VENTOS (small extension) and WRF. The black line shows the West Portuguese coast.

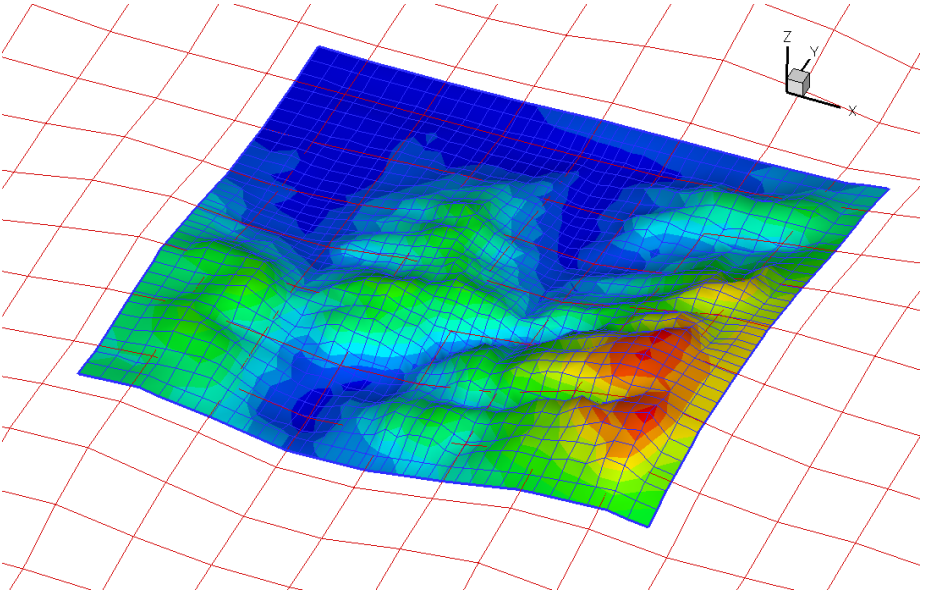


Fig. 5. Meshes and topography used by both codes

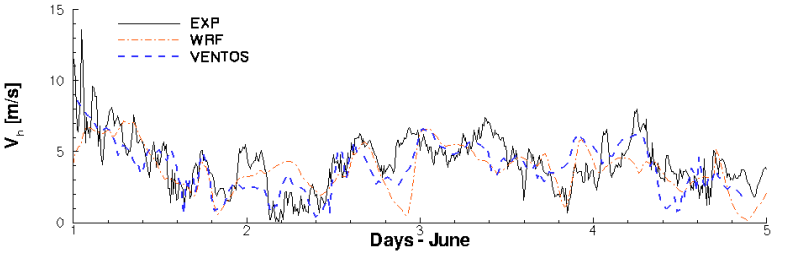


Fig. 6. Time series of the horizontal velocity for the June occasion

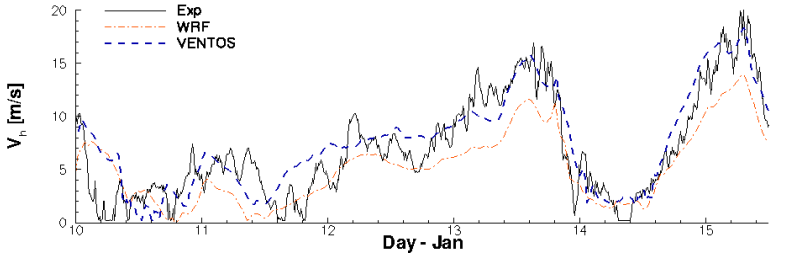


Fig. 7. Time series of the horizontal velocity for the January occasion

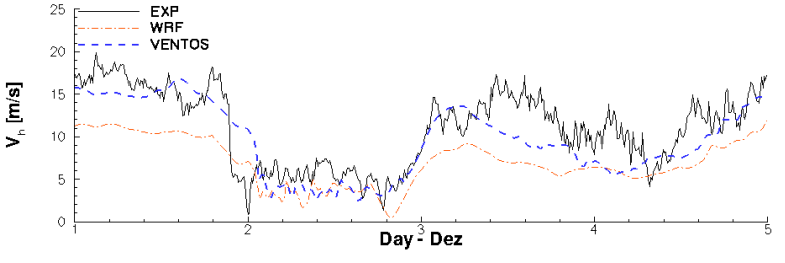


Fig. 8. Time series of the horizontal velocity for the December occasion

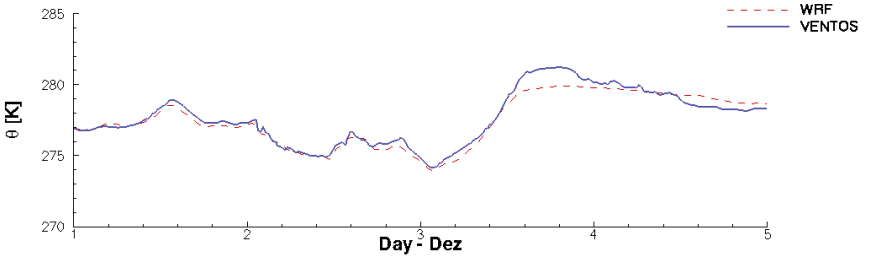


Fig. 9. Time series of the potential temperature for the December occasion

Table 2. RMS errors in m/s for the predictions at PORT267 for the 3 occasions under study

Month	VENTOS	WRF
June	1.49	1.59
January	2.17	2.66
December	2.62	4.35

Figure 6 shows the time series of V_h for the warmer occasion under study, 10 to 15 of June. For this period the wind speed was not very height and so thermal effects are more prone the determine the flow behaviour. In this case, the VENTOS results were not a significant improvement over the WRF results. This is partially explained by the simpler thermodynamics and heat-transfer models used in VENTOS.

For the winter occasions, figures 7-8, the mean observed wind speed is higher and the overall improvement introduced by VENTOS is very noticeable. In the windiest period of the study, the first days of December, the improvement was quite significant, when the WRF results were showing wind speeds almost 40% lower than the experimental results.

The potential temperature predicted by VENTOS for December follows very well the WRF results, as can be seen in figure 9. This shows that the thermodynamics and heat-transfer models used in VENTOS are well suited for the application in mind, i.e. the wind power prediction, that deals mainly with the operation of wind turbines for velocities above nearly 5 m/s.

The rms errors in m/s for the 3 periods under study are presented in table 4.2. For the more interesting case, from the wind power point of view, the December period, VENTOS reduced the error at PORT267 1.7 times.

5 Conclusions

The parallel version of the VENTOS CFD code was developed with the aim of producing short term weather prediction. The parallelisation of a CFD code was performed using a domain decomposition strategy, where the physical simulation domain, discretised by a mesh of control volumes with a central node, was decomposed into several sub-domains, each being calculated in a dedicated processor. The Message Passing Interface library (MPI) was used to implement the communication between processes. High parallel efficiencies were obtained ($> 80\%$) even for 20 processors. The parallelisation introduced some decoupling between sub-domains which can degrade the converge rate for certain cases. Nevertheless, efficiencies of 70% are still obtained. The adopted strategy and its numerical implementation permitted sufficiently faster execution times to enable true predictions using the current CFD code.

A test case using a real flow over the wind farm of Mendoiro/Bustave, in Portugal, showed that the coupling procedure can improve the mesoscale results.

The improvement was more noticeable in windy conditions, the preferred situation for the application in mind, the wind power production. In the better case, a reduction of 1.7 $\times$ in the error of the mesoscale results were obtained.

Acknowledgements. Carlos Silva Santos would like to ackowledge the support of FCT, Fundação para a Ciência e Tecnologia, through grant SFRH/BPD/18655/2004, co-financed by program POCI 2010 and the European Union.

References

1. Castro, F.A.: Numerical Methods for the Simulation of Atmospheric Flows over Complex Terrain (in Portuguese). PhD thesis, Faculty of Engineering of Porto (1997)
2. Castro, F., Palma, J., Lopes, A.S.: Simulation of the askervein flow. part 1: Reynolds averaged Navier–Stokes equations (k-ϵ turbulence model). Boundary-Layer Meteorology (2003)
3. Jones, W.P., Launder, B.E.: The prediction of laminarization with a two-equation model of turbulence. International Journal of Heat and Mass Transfer 15, 301–314 (1972)
4. Beljaars, A.C.M., Walmsley, J.L., Taylor, P.A.: A mixed spectral finite-difference model for neutrally stratified boundary-layer flow over roughness changes and topography. Boundary-Layer Meteorology 38, 273–303 (1987)
5. Knupp, P., Steinberg, S.: Fundamentals of grid generation. CRC Press, Boca Raton (1994)
6. Ferziger, J.H., Perić, M.: Computational Methods for Fluid Dynamics, 3rd edn. Springer, Heidelberg (2001)
7. Rhie, C.M., Chow, W.L.: Numerical study of the turbulent flow past an airfoil with trailing edge separation. AIAA Journal 21, 1525–1532 (1983)
8. Miller, T.F., Schmidt, F.W.: Use of a pressure-weighted interpolation method for the solution of the incompressible Navier-Stokes equations on a nonstaggered grid system. Numerical Heat Transfer 14, 212–233 (1988)
9. Ferziger, J.H., Perić, M.: Computational Methods for Fluid Dynamics. Springer, Heidelberg (1996)
10. Durst, F., Schafer, M.: A parallel block-structured multigrid method for the prediction of incompressible flows. Int. J. Numer. Meth. Fl 22, 549–565 (1996)

High Performance Computing for Eigenvalue Solver in Density-Matrix Renormalization Group Method: Parallelization of the Hamiltonian Matrix-Vector Multiplication

Susumu Yamada, Masahiko Okumura, and Masahiko Machida

CCSE, Japan Atomic Energy Agency, 6-9-3 Higashi-Ueno,
Taito-ku, Tokyo, 110-0015, Japan
&
CREST(JST), 4-1-8 Honcho, Kawaguchi, 330-0012, Japan
{yamada.susumu,okumura.masahiko,machida.masahiko}@jaea.go.jp

Abstract. The Density Matrix Renormalization Group (DMRG) method is widely used by computational physicists as a high accuracy tool to obtain the ground state of large quantum lattice models. Since the DMRG method has been originally developed for 1-D models, many extended method to a 2-D model have been proposed. However, some of them have issues in term of their accuracy. It is expected that the accuracy of the DMRG method extended directly to 2-D models is excellent. The direct extension DMRG method demands an enormous memory space. Therefore, we parallelize the matrix-vector multiplication in iterative methods for solving the eigenvalue problem, which is the most time- and memory-consuming operation. We find that the parallel efficiency of the direct extension DMRG method shows a good one as the number of states kept increases.

Keywords: Parallel and distributed computing, DMRG method, matrix-vector multiplication, eigenvalue problem, quantum lattice systems.

1 Introduction

Quantum lattice systems, e.g. Heisenberg model[1] and Hubbard model[2,3], have attracted a tremendous number of physicists since the systems exhibit a lot of interesting phenomenon such as High-T_c superconductivity. In order to understand the systems, some computational methods have been proposed. The most accurate one of them is the exact diagonalization method, which solves the ground state (the smallest eigenvalue and the corresponding eigenvector) of the Hamiltonian matrix derived from the systems. We have actually parallelized the exact diagonalization method and obtained some novel physical results[4,5,6,7]. However, the dimension of the Hamiltonian matrix for the exact diagonalization method increases almost exponentially with the number of the lattice sites.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 39–45, 2008.

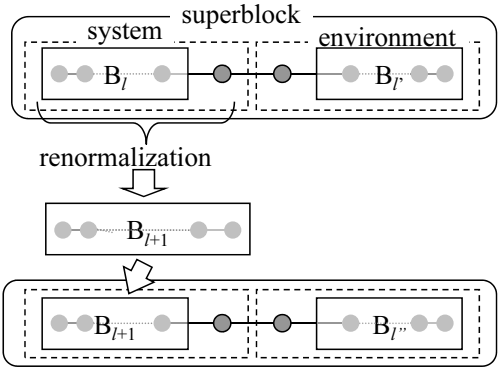


Fig. 1. A schematic figure of the renormalization scheme of DMRG method for a 1-D lattice model. The "superblock" is composed of the "system" and the "environment". The rectangles inside of the above superblock indicate the blocks which have l and l' lattice sites, and the dark circles present single sites. New block B_{l+1} is formed by renormalizing the left block B_l and the left single with keeping the number of the states in the system block.

Thus, the limit of the simulation on a supercomputer with a terabyte memory system is an about-20-site system[6,7].

In order to overcome the memory-size explosion problem, the Density Matrix Renormalization Group (DMRG) method, which keeps the number of the relevant quantum states constant by renormalizing the states of the previous step on enlarging the system (see Fig. 1), has been proposed by S. R. White[8,9]. The DMRG has originally been developed for 1-leg (1-D) lattice models, and a lot of problems for 1-D quantum lattice systems has been resolved. The DMRG method can be directly extended to an s-leg (2-D) model as depicted in Fig. 2. The extension strategy is promising for the excellent accuracy and the good convergence property. However, the method leads to a large amount of memory consumption, since the maximum number of the states required in the direct extension DMRG algorithm is given as $16^s m^2$ for the s-leg Hubbard model per block, in which s and m are the number of the sites in the rung direction and the number of states kept, respectively. Although the degree of freedom practically decreases by eliminating physically irrelevant states, it is clear that even a slight increment of the legs gives rise to an exponential growth of the state number. Therefore, the previous 2-D DMRG method have adopted "multichain algorithm", as depicted in Fig. 3, since its memory space is principally comparable to the 1-D case. However, there remain unsolved issues in terms of its accuracy[10,11]. Thus, we study the parallelization technique for the direct extension of DMRG method to s-leg models on a distributed-memory parallel computer, which totally has a huge memory system.

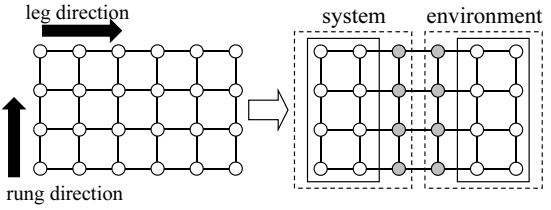


Fig. 2. A superblock configuration in the direct extension of DMRG to a 4-leg model

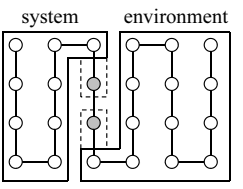


Fig. 3. A superblock in the multichain algorithm

2 Density Matrix Renormalization Group Method

2.1 Algorithm

In the DMRG method, the whole system of the lattice model is called "superblock", and the "superblock" is split into two blocks, which are called "system block" and "environment block" (see Fig. 1). The procedure of the DMRG method for 1-D model is described as follows:

1. Form the Hamiltonian matrix form the superblock and find the ground state of the matrix.
2. Form the density matrix for the system block using the ground state.
3. Solve the eigenvalue problem for the density matrix, and discard all expect for the largest m eigenvalues and the corresponding eigenvectors. Here, m means the number of states kept per block.
4. Form the new system block, which is the previous system block plus one site, using the eigenvectors, and construct the new superblock out of the new system block and the relevant environment block.

In the direct extension DMRG for the s-leg model, a new system block is formed with the previous system block and s sites. In this case, the dimension of the Hubbard Hamiltonian matrix for the new superblock is $16^s m^2$. Although the dimension can decrease by eliminating the irrelevant states, the calculation for solving the ground state of the Hamiltonian matrix is the most time- and memory-consuming operation. Since the Hamiltonian matrix is a sparse symmetric matrix, an iterative method, such as the Lanczos method [12] and the conjugate gradient method [13,14], are utilized for solving the ground state. For these iterative methods, the most difficult part is the parallelization of the multiplication of the Hamiltonian matrix and a vector. Therefore, we focus on the parallelization for the multiplication in the following.

2.2 Parallelization of Matrix-Vector Multiplication in DMRG

Each block of the superblock for the direct extension 2-D DMRG method is called "block 1", "block 2", "block 3", and "block 4" from the left, and state

of the "block j" is represented as i_j. We decompose the superblock into the left block, the central block, and the right block as Fig. 4, and the Hamiltonian matrices for the three blocks are represented as H_l, H_c, and H_r, respectively. The Hamiltonian matrix for the superblock is decomposed into three matrices, i.e. H_1, H_2, and H_3, which correspond to the states i_1 and i_2, i_3 and i_4, and i_2 and i_3, respectively. Thus, the matrix-vector multiplication Hv is partitioned as

$$Hv = H_1 v + H_2 v + H_3 v. \tag{1}$$

Here, let the $((i_3 - 1)m^2 n + (i_4 - 1)mn + (i_2 - 1)m + i_1)$-th element of the vector v, which corresponds to the $|i_1 i_2 i_3 i_4 >$ state, transform into the element $((i_2 - 1)m + i_1, (i_3 - 1)m + i_4)$ of a matrix V (see Fig. 5), in which $n = 4^s$ in s-leg ladder Hubbard model. Then, the first two multiplications of (1) can be transformed into matrix-matrix multiplications as

$$H_1 v \rightarrow H_l V,$$
$$H_2 v \rightarrow V H_r^T.$$

In addition, when the element of the vector v is transformed into the element $((i_3 - 1)n + i_2, (i_4 - 1)m + i_1))$ of a matrix V_c (see Fig. 5), the last operation of (1) is also transformed into a matrix-matrix multiplication as

$$H_3 v \rightarrow H_c V_c.$$

We find that the matrix-vector multiplication Hv can be decomposed into three matrix-matrix multiplications. It is expected that the multiplications can be effectively parallelized by partitioning the matrices V and V_c. The parallel computation for the multiplication Hv is realized by four calculation stages and four communication ones as follows:

CAL 1: $W_1^c = H_l V^c$,
COM 1: communication for transforming V^c to V^r,
CAL 2: $Z_2^r = V^r H_r^T$,
COM 2: communication for transforming Z_2^r to W_2^c,
COM 3: communication for transforming V^c to V_c^c,
CAL 3: $Z_3^c = H_c V_c^c$
COM 4: communication for transforming Z_3^c to W_3^c,
CAL 4: $W^c = W_1^c + W_2^c + W_3^c$,

where the superscription c and r denote the columnwise and rowwise partitioning in matrix data distribution, respectively. Although the matrices V and V_c are originally dense matrices, the elements corresponding to the irrelevant states are eliminated. Therefore, the matrices should be partitioned in consideration of the arrangement of the relevant elements.

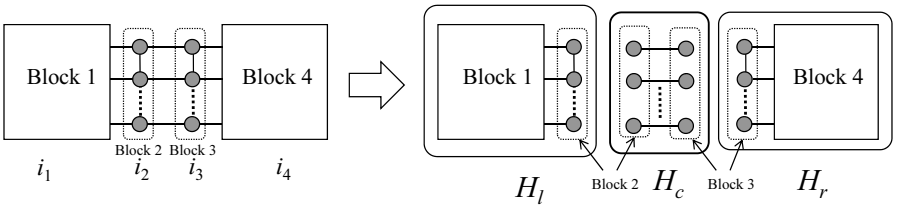


Fig. 4. A definition for the Hamiltonian matrices H_l, H_r, and H_c. The superblock as shown in the left side is decomposed into three blocks, i.e. the left block, the right block, and central block, as shown in the right side. The matrix H_l, H_r, and H_c are the Hamiltonian matrices for the left block, the right block, and central block, respectively.

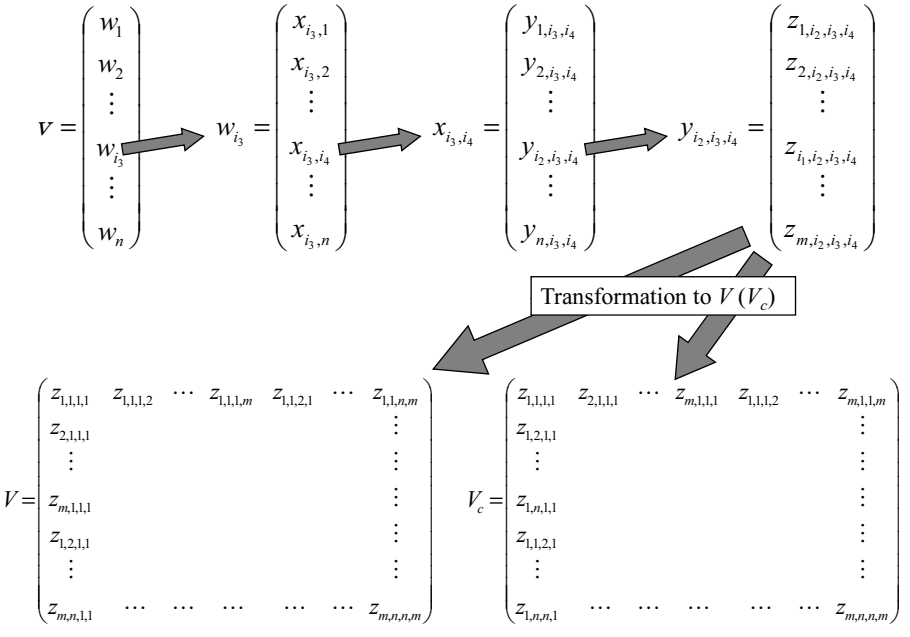


Fig. 5. A way to transform the vector v to the matrices V and V_c. The vector v is hierarchically arranged as the upper of the figure. The element $z_{i_1 i_2 i_3 i_4}$is arranged in the matrix as this figure.

2.3 Data Locality

In the Hubbard model, there is no correlation among states with different number of spins. Therefore, the three matrices H_l, H_c, and H_r, which are the Hamiltonian matrices for the left block, central block, and right block, respectively, become the block diagonal matrices, if the elements are rearranged by the number of spins. The rearrangement improves the data locality, which strongly influences the performance on a scalar computer.

3 Numerical Experiment

We examine the performance of the direct extension 2-D DMRG method on SGI Altix3700Bx2 in Japan Atomic Energy Agency. A test example is 4-leg (10×4-site) Hubbard model with 38 fermions (19 up-spins, 19 down-spins). Figure 6 shows a relationship between the number of states kept m and the elapsed time. We find that as the number of processors increases from 32 to 128, the elapsed time is reduced to be about one half, when m is about larger than 128. The result shows that the present parallelization scheme is promising, since the obtained ground state approaches to the true ground state with increasing the number of the states kept m.

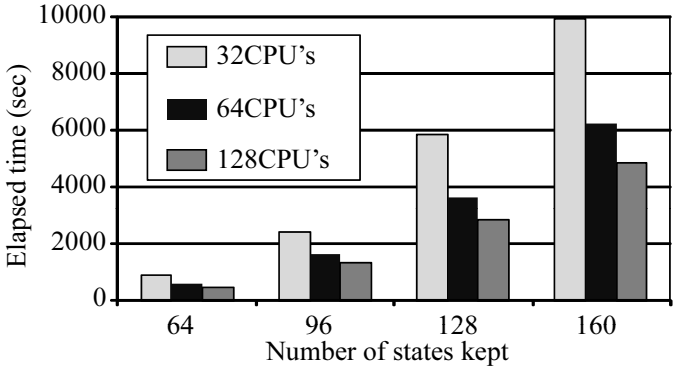


Fig. 6. Relationship between the number m of states kept and the elapsed time of the direct extension DMRG method for 10×4-site Hubbard model on SGI Altix 3700Bx2

4 Conclusion

In order to execute the direct extension of DMRG method to s-leg model on distributed-memory parallel computers, we parallelized and tuned mainly the matrix-vector multiplication in solving the eigenvalue problem for the Hamiltonian matrix, which is the most time- and memory-consuming operation. Consequently, we found that the parallelization efficiency increases with increasing the number of the states kept. Therefore, we believe that the direct extension 2-D DMRG method is a promising tool to explore large quantum 2-D lattice systems.

Acknowledgments. This research was supported by Strategic International Cooperative Program, Japan Science and Technology Agency (JST). This was also partially supported by Grant-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology of Japan (Grant No.20500044).

References

1. Kennedy, T., Nachtergaele, B.: The Heisenberg Model - a Bibliography (1996), http://math.arizona.edu/~tgk/qs.html
2. Rasetti, M. (ed.): The Hubbard Model: Recent Results. World Scientific, Singapore (1991)
3. Montorsi, A. (ed.): The Hubbard Model. World Scientific, Singapore (1992)
4. Machida, M., Yamada, S., Ohashi, Y., Matsumoto, H.: Novel superfluidity in a trapped gas of Fermi atoms with repulsive interaction loaded on an optical lattice. Phys. Rev. Lett. 93, 200402 (2004)
5. Machida, M., Yamada, S., Ohashi, Y., Matsumoto, H., Machida, et al.: Reply. Phys. Rev. Lett. 95, 218902 (2005)
6. Yamada, S., Imamura, T., Machida, M.: 16.447 TFlops and 159-Billion-dimensional Exact-diagonalization for Trapped Fermion-Hubbard Model on the Earth Simulator. In: Proc. of SC 2005 (2005), http://sc05.supercomputing.org/schedule/pdf/pap188.pdf
7. Yamada, S., Imamura, T., Kano, T., Machida, M.: High-Performance Computing for Exact Numerical Approaches to Quantum Many-Body Problems on the Earth Simulator. In: Proc. of SC 2006 (2006), http://sc06.supercomputing.org/schedule/pdf/gb113.pdf
8. White, S.R.: Density Matrix Formulation for Quantum Renormalization Groups. Phys. Rev. Lett. 69, 2863–2866 (1992)
9. White, S.R.: Density-matrix algorithms for quantum renormalization groups. Phys. Rev. B 48, 10345–10355 (1993)
10. Hager, G., Wellein, G., Jackemann, E., Fehske, H.: Stripe formation in dropped Hubbard ladders. Phys. Rev. B 71, 75108 (2005)
11. Noack, R.M., Manmana, S.R.: Diagonalization and Numerical Renormalization-Group-Based Methods for Interacting Quantum Systems. In: Proc. of AIP Conf., vol. 789, pp. 91–163 (2005)
12. Cullum, J.K., Willoughby, R.A.: Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Theory, vol. 1. SIAM, Philadelphia (2002)
13. Knyazev, A.V.: Preconditioned eigensolvers - An oxymoron? Electronic Transactions on Numerical analysis 7, 104–123 (1998)
14. Knyazev, A.V.: Toward the optimal eigensolver: Locally optimal block preconditioned conjugate gradient method. SIAM J. Sci. Comput. 23, 517–541 (2001)

Tunable Parallel Experiments in a GridRPC Framework: Application to Linear Solvers

Yves Caniou[1,⋆,⋆⋆], Jean-Sébastien Gay[2,⋆,⋆⋆], and Pierre Ramet[3,⋆,⋆⋆⋆]

[1] LIP-ÉNS de Lyon, Université Claude Bernard de Lyon
yves.caniou@ens-lyon.fr
[2] LIP-ÉNS de Lyon
Jean-Sebastien.Gay@ens-lyon.fr
[3] LABRI, Université Bordeaux 1
ramet@labri.fr

Abstract. The use of scientific computing centers becomes more and more difficult on modern parallel architectures. Users must face a large variety of batch systems (with their own specific syntax) and have to set many parameters to tune their applications (*e.g.,* processors and/or threads mapping, memory resource constraints). Moreover, finding the optimal performance is not the only criteria when a pool of jobs is submitted on the Grid (for numerical parametric analysis for instance) and one must focus on the wall-time completion. In this work we tackle the problem by using the DIET Grid middleware that integrates an adaptable PASTIX service to solve a set of experiments issued from the simulations of the ASTER project.

Keyword: Grid computing, Sparse linear solver, Performance prediction, Application specific plug-in scheduling.

1 Introduction

Parallel computing and the design of high-performance codes to be executed on today's computing platforms are one of the major research activities in computer science. Such architectures are now parallel computers organized as a large network of SMP nodes and/or Grid platforms. On an other hand, solving large sparse systems of linear equations is a crucial and time-consuming step, arising in many scientific and engineering applications. Consequently, many parallel techniques for sparse matrix factorization have been studied and implemented.

In the context of the ASTER project (*Adaptive MHD Simulation of Tokamak ELMs for* ITER), we develop and implement methods to improve the simulation of MHD instabilities that are needed to evaluate mechanisms to control the energy losses observed in the standard tokamak operating scenario (ITER). To resolve a wide range of timescales, a fully implicit time evolution scheme is used; this leads to a large sparse matrix system

⋆ This work is supported by the REDIMPS project JST-CNRS.
⋆⋆ This work is supported by the LEGO project ANR-05-CIGC-11.
⋆⋆⋆ This work is supported by the ASTER project ANR-06-CIS-1 and SOLSTICE project ANR-06-CIS-10.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 46–52, 2008.

to be solved at every time step. To reduce the large memory requirements of the sparse matrix solve, the PASTIX library [7] is being extended to support preconditioners using incomplete factorization.

One of the aim of the ASTER project is to define the choice of optimal parameters of the solver on a collection of test cases and to analyze the efficiency and convergence for both parallel and sequential implementations [4]. Then, we must perform an exhaustive set of experiments, whose objective is to collect the benchmark results induced by this parametric analysis.

We address the efficiency problem with a Grid architecture relying on the DIET [3] Grid middleware. The proposed solution integrates the development of a DIET client/server which gives access to the PASTIX service over the Grid, a transparent batch parallel job submission mechanism to address batch systems heterogeneity as well as mechanisms to obtain static *and* dynamic information on the resources of a site, a parallel job tuning to address the moldability of PASTIX (the possibility to set the number of processors to use at launch time), and a distributed application-specific scheduler.

2 Related Work

The TLSE project [1] (Test for Large Systems of Equations) aims to provide an expert Grid system, particularly to evaluate sparse direct solvers. TLSE relies on the DIET Grid middleware to submit the computing analysis on the Grid. The context of work is different than the one in this paper, because submission of parallel jobs was not available within the DIET API and such had to be done case by case (forks or batch scripts had to be hard coded if used), and iterative solvers functionalities (incomplete factorization for instance) cannot be taken into account. In any case, the platform will not allow performance predictions and approximate wall-time. Furthermore, we want to take advantage of the moldability of PASTIX parallel jobs, which can be tuned accordingly, for example to benefit of the maximum idle resources on a site.

3 Presentation of PASTIX

PASTIX [2] is a scientific library that provides a high performance parallel solver for very large sparse linear systems based on block direct and block ILU(k) iterative methods.

The PASTIX library uses the graph partitioning and sparse matrix block ordering package `Scotch` [8]. PASTIX is based on an efficient static scheduling and memory manager by taking into account very precisely the computational costs of the BLAS primitives, the communication costs and the cost of local aggregations.

In the context of SMP node architectures, we fully exploit shared memory advantages. A relevant approach is then to use an hybrid MPI-thread implementation. This not yet explored approach in the framework of direct solver aims at efficiently solving 3D problems with much more than 10 millions of unknowns. We have shown that this

[1] `http://gridtlse.org/`
[2] `http://pastix.gforge.inria.fr`

approach allows a great reduction of the memory required for communications [6]. Hybrid MPI-thread batch scheduling is then crucial to solve large problems even if those requirements are often difficult to set on scientific computing center.

4 Presentation of DIET

DIET is a GridRPC middleware relying on the client/agent/server paradigm. A **client** is an application which needs a computing service. The **agent**, which can be extended as a hierarchy of agents, has the knowledge of several servers. The distributed scheduler embedded in each agent chooses the best computing resource for the execution of a given request considering a given metric. The **server** is a daemon running on the computing resource. The server gives performance estimations to its agent and launches a service each time requested by the client.

The mechanism to execute a request is shown in Figure 5: when an agent is contacted by a client who wants to solve a problem (a), the request follows down the hierarchy of agents to servers (b and c). They answer back performance information (c and b) which is used up in the hierarchy to determine which one suits the best the resolution of the service (b). The identity of the server is given to the client, who contacts the server and sends its data (f). Once the computation is finished, results are transfered back to the client.

5 Architecture of the Proposed Solution

The architecture of the solution is schemed in Figure 5. There are four main parts. In the reverse order of the process of a request: a script parameterized with correct values concerning the number of processors used in regard to both PASTIX and to the batch scheduler has to be created. This means that the server can access the number of available idle resources on the site through the batch scheduler (d); integrate a correct knowledge to decide how many of them the PASTIX service will use (d); create and transparently submit the batch script ((d) and (e)); higher in the hierarchy, the request is processed by the hierarchical scheduler, which is specifically designed for the PASTIX service (b). All this steps are described in this section.

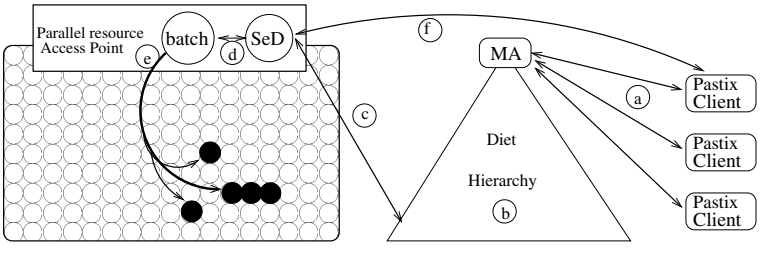


Fig. 1. Architecture of the proposed solution

5.1 Improvements Realized in the DIET Grid Middleware

Re-designing the Implementation of DIET Servers

A server is now either a `SERIAL`, `PARALLEL` or a `BATCH` server. A `SERIAL` server is the previous and unique kind of server available in DIET. It launches the resolution of a service by forking the function which realizes the solve. It is still the default behavior if none is given in the server code. A `PARALLEL` server reads in a file the name of the resources that it manages, and knows consequently their number. It launches a script provided by the SED programmer (generally a MPI script) where the resources and their number can be used with the help of meta-variables. It implied additional developments because the script is forked but its execution has still to be controlled for asynchronous mode, to advertise the client when the job is terminated, or simply to know when transferring back the results. A `BATCH` server reads the name of the batch system in the server configuration file, and can submit consequently adaptable batch scripts provided by the SED programmer. In that case, the SED requests the batch system every 30 seconds to control the state of the job. For the moment, OARv1.6 and Loadlever batch systems are supported.

A service is now declared and registered as *sequential* or *parallel*. This has three main purposes: 1) A `PARALLEL` or `BATCH` server can provide two different implementations, one sequential and the other parallel, with the same name. Hence, a user can request explicitly for one kind of service to explore some speed-up study, or the DIET server can dynamically choose which one to execute depending on system performances. Then, in that case, the user does not even now the nature of the resolution nor the machine that has performed the resolution but only gets the results when the job is finished. 2) The service and some of its characteristics (like being parallel or sequential) are registered in all the components from the server to the root agent (the Master Agent) in the hierarchy. By default, if no constraint is specified by the user, DIET tries to answer to the request with the best server (according to some metrics) able to solve the service corresponding to the service name, and arguments number and type given in the request. But if the client specifies the type of the service that he wants, then the request is only forwarded down in the hierarchy to components that are aware of this kind of service, making the scheduling process lighter and the latency smaller. 3) When arriving at the last local agent, if there is no constraint on the request and if the server can solve both kind of services, the request is duplicated when submitted to the server, each one having the `SERIAL` or `PARALLEL` characteristic set. Hence, one can achieve traditional (but surely inefficient) Round-Robin between services on all the platform as a default scheduling policy.

An Extended API to Address Batch Systems Performance Estimations

DIET makes use of a performance module called CoRI (Collector of Resource Information). CoRI proposes a generic API which lets the user get some given information on the system by transparently transferring the request to a software or a built-in tool. It is also used to set information that are transferred back to the agent, for application specific scheduling.

We have integrated a submodule which can get information through batch schedulers. For now, only the necessary information for the work of this paper can be obtained, namely the number of processors of the parallel resource and the number of idle processors available in the system. Work has been made for new functionalities to be added easily in the code and in the API.

A New API to Interface DIET and Batch Schedulers

We have extended the DIET API in order to provide means to submit a draft script constructed by the SED programmer that DIET completes and submits to the system with correct batch related information set (if `PARALLEL` or `BATCH`). Hence, DIET transparently manages batch and parallel submissions. In order for the user to be able to use system dependent information, DIET proposes a set of meta-variables that the SED programmer can use in the draft script. They describe the system and are replaced by the SED at launch time, The script is then submitted with the additional corresponding batch syntax. Hence, the submission is conducted transparently for the client/server programmer. This gives a higher level and much ease to the server programmer to design a single generic server working on different batch systems. A small common set of information can be used (number of processor, name of the frontal for `ssh` copies, etc.) and further work has to be performed.

5.2 Interfacing PASTIX and DIET

Interfacing PASTIX and DIET occurs at three very different levels: first, we need to use DIET functionalities to question the batch system about its load. We need as well a PASTIX performance prediction to decide how many threads per reserved processors will be used, in order to fully benefit from the moldability of PASTIX jobs; second, a DIET client/server has to be designed with the DIET batch API, in order to propose the PASTIX service to Grid users; third, due to the nature of the analysis, we focus on cycle stealing. Then the scheduling metric is based on idle resources and an application specific plug-in scheduler has to be defined.

Performance Predictions

Since the ordering step and the block symbolic factorization can be pre-calculated for each problem, PASTIX is able to quickly estimate the execution time in function of the number of processors (threads) before factorizing and solving the system. We can notice that the size of the block symbolic matrix, used to compute the prediction, is small. Moreover, at the same time, the exact memory requirement for each processor can be computed. The script can then be built by DIET to fit the best PASTIX requirements and accordingly to system performances (*e.g.,* number of processors, walltime).

Designing the DIET Client/Server to Provide the PASTIX Service

We have used the new DIET batch API to write the PASTIX client/server, in order to be able to submit transparently to clusters managed with OAR and to an AIX parallel

machine managed with Loadleveler. Nonetheless, writing the DIET client/server is very similar to the work in [1], except that meta-variables are used in a draft script and it is launched with a special call, which completes the batch script and launches the script in place of both the client and server programmer to the batch scheduler. The draft script built by the DIET server takes into account the values given by the performance prediction to reserve the correct number of processors (threads) for the correct duration.

Scheduling PaStiX Requests in DIET

Because we plan an extensive analysis, we need the maximum available resources. In this paper, we consider a scheduling based on the availability of resources. The work is performed using the new CoRI batch submodule to get the corresponding information on the system and the plug-in scheduler functionality of DIET [2]. At each step in the hierarchy, the aggregation method sorts the servers by the maximum available idle resources.

6 Conclusion

Concerning PaStiX, all the results and data will be collected in order to select the parameters of the solver that best fit the simulations of MHD instabilities. Some improvements are currently developed into the solver to take care of NUMA (Non-Uniform Memory Access) effects. This will induce some new constraints regarding the mapping of resources during batch reservations on such architectures.

Concerning DIET, we will extend the number of recognized batch with OpenPBS and SGE batch reservation systems. Furthermore, we will provide more functionalities regarding batch integration: an improved performance prediction for given batch systems (collecting information *and* predicting system behaviors for example with the help of Simbatch [5]), as well as better autoconfiguration tools on the server side, to automatically discover batch queues and their respective information. These will lead to a better information quality, which will be used in the DIET scheduling.

We plan to use these improvements in a future work concerning the resolution of a set of experiments where the memory has to be taken into account in the scheduling process as well as when solving the problem. Furthermore, the schema of this set of experiments can be represented as a workflow whose branches can be pruned depending on temporary results. Scheduling algorithms have then to be studied and tested.

The DIET extensions developed in this work are directly integrated in the internals of DIET, and will be integrated in the LEGO demonstrator for the evaluation of the project. Furthermore, it will be adapted when possible into the next evolutions of TLSE.

References

1. Caniou, Y., Caron, E., Courtois, H., Depardon, B., Teyssier, R.: Cosmological simulations using grid middleware. In: Fourth High-Performance Grid Computing Workshop. HPGC 2007, Long Beach, California, USA, March 26, 2007. IEEE, Los Alamitos (2007)
2. Caron, E., Chis, A., Desprez, F., Su, A.: Design of plug-in schedulers for a gridrpc environment. Future Generation Computer Systems 24(1), 46–57 (2008)

3. Caron, E., Desprez, F.: Diet: A scalable toolbox to build network enabled servers on the grid. International Journal of High Performance Computing Applications 20(3), 335–352 (2006)
4. Czarny, O., Huysmans, G., Hénon, P., Ramet, P.: Improvement of existing solvers for the simulation of mhd instabilities. In: Numerical flow models for controlled fusion, Porquerolles, France (April 2007)
5. Gay, J.-S., Caniou, Y.: Étude de la précision de Simbatch, une API pour la simulation de systémes batch. RSTI - Techniques et Sciences Informatiques 27, 373–394 (2008)
6. Hénon, P., Ramet, P., Roman, J.: On using an hybrid mpi-thread programming for the implementation of a parallel sparse direct solver on a network of smp nodes. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 1050–1057. Springer, Heidelberg (2006)
7. Hénon, P., Ramet, P., Roman, J.: On finding approximate supernodes for an efficient ilu(k) factorization. Parallel Computing (accepted, 2007)
8. Pellegrini, F.: Scotch 4.0 User's guide. Technical report, Inria Futurs (April 2005)

The Rise of the Commodity Vectors

Satoshi Matsuoka

Tokyo Institute of Technology
National Institute of Informatics / JST CREST
matsu@is.titech.ac.jp

Abstract. Commodity clusters revolutionized high performance computing in a fundamental way since its first inception, and now now dominate many of the world's premiere supercomputers on the Top500, growing to the scale of over 10,000 CPU cores and beyond. Still,"classical" specialized vector supercomputers still remain to be sold and facilitated, especially at high end of the market, largely due to the nature of some of the HPC workloads still requiring the computing power of vectors, in areas such as CFD, FEM with kernels such as FFT, characterized as mostly large-scale irregular sparse codes Finally, however, commoditization of vector computing is on the rise, lead by multimedia application requirements, and spurred many architectures to arise such as GPUs and the Cell processor. But various problems still remain by which we cannot claim with 100% confidence that commodity vectors are here to stay in the HPC space. Research and development has to be conducted at various degrees of intensity to utilize the new breed of commodity vector hardware to their fullest capabilities, just as various research were needed to harness the power and the scalability of commodity clusters. In the talk I will outline some of the details, and our recent research endeavors aimed at solving the various issues.

1 Introduction—The Rise of the Commodity Vectors

Commodity clusters revolutionized high performance computing in a fundamental way, Since its first inception (Wigraf) in 1994, it quickly spread and rode the rapid technology curve advances of commodity microprocessors, and now dominate many of the world's premiere supercomputers on the Top500 [1], growing to the scale of over 10,000 CPU cores and beyond.

The biggest cultural effect that were given rise by cluster computing was to bring HPC to the mainstream--- the building block components were those of commodity in nature, especially in the (albeit high-end) PC desktop & server ecosystem, and the architectural research on constructing a supercomputer now rested on how the components are pieced together using innovative software techniques. This is in stark contrast to the old days of customized supercomputers of the past---the Crays / NEC SXs / Fujitsu VPPs / ETA10 / … etc., that star-studded the supercomputing arena only a decade earlier.

Commoditization is very important from the industry ecosystem viewpoint, both for economy and sustainment of the infrastructure, as supercomputing now cannot be alienated from rest of the IT, both from a financial point of view as well as from a

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 53–62, 2008.

technical point of view. For example, software compatibility and longevity are very important, from the software vendor's standpoint in market penetration and customer retention, and from the user's standpoint in availability/longevity, as well as continuum with the now complex software R&D environment he would be using every day with sophisticated GUIs instead of the arcane half-duplex TTY systems (or even punch cards in the good old days.)

Still, circa early 2008, one also notices that ``classical'' specialized vector supercomputers, and non-commodity CPUs that emulate them to some extent, still remain to be sold and facilitated, especially at high end of the market, by major supercomputing centers. These include the Cray Black Widow, the NEC SX-8/8R/9, and the high-end IBM Power servers. The existence is largely due to the nature of some of the HPC workloads still requiring the computing power of vectors, mostly those that require high bandwidth and amenable to vectorization. These include areas such as CFD, FEM with kernels such as FFT, characterized as mostly large-scale irregular sparse codes sometimes with serialization bottlenecks. One could say that, irregular sparse codes and their kernels are the last stronghold of vector supercomputing market, as performance parity could occur for those types of applications due to higher efficiency, and as a result, may lead to overall advantage for the classical vectors in the overall price, facility requirements, or application porting (for codes that had been optimized for vectors).

Finally, however, starting from a few years ago, commoditization of vector computing is on the rise, driven by the needs of multimedia, gaming, video, and other applications to offer richer UI experiences. Now, such capabilities are starting to be applied to non-graphical/media technical computing. This is synonymous to the first Beowulf cluster in 1994 where commodity desktop PC processors (Intel 486 CPUs at the time) were utilized to construct the first computing cluster. Despite that some of the earlier attempts especially. with GPUs had less than stellar performance, and/or the user having to deal with arcane programming model based on graphical pipelines, inherent advances to enrich the user experiences had lead to generalization of the overall architecture to be significantly resemble traditional many-core, multithreaded SIMD CPU in order to cope with very sophisticated rendering algorithms. As a result, recent hardware and software advances thereof are finally making the system to be tractable for a non-graphics layman in scientific computing, and the performance advantage potentials that are beginning to find substantive traction and impetus in the community. A recent IEEE article[2] gives an excellent overview of the history and the status quo of GPU computing, especially focusing on molecular dynamics that the one of the authors' group had been working on in the context of Folding@Home[3] and other apps. Similar can be said for non-graphical but multimedia-oriented CPUs such as the CELL Broadband Engine[4].

There are less commoditized but still fairly inexpensive and general-purpose SIMD-Vector accelerators on the market, such as the ClearSpeed Accelerator board[5], the ones in particular which are housed in Tokyo Tech.'s supercomputer TSUBAME, representing almost half of the peak computing power provided by our Global Scientific Information and Computing Center (GSIC). ClearSpeed has been the driving force of TSUBAME's ability to retain the top performing position as the Japan's No. 1 supercomputer on the Top500, due to several innovative works in heterogeneous Linpack algorithm that we had developed that allowed increase of

performance over four consecutive rankings, the first of its kind in the world[6]. Still, one could claim that ClearSpeed is situated in the realm of commodity space, as it employs standard interfaces to the PC (PCI-X and PCI-E), and as such could be added onto any commodity desktop PC---and its price range is on the high end of commodity platforms as well, in the low thousands of dollars as of 2008.

Some may also claim that we had gone the way before---certainly we remember the old days of Weitek vector add-ons for early RISC based MPPs such as the Thinking Machine's CM-5, Meiko CS-2, etc. There had also been dedicated accelerators that were more generic, such as Riken's Grape series such as the MD-Grape 2[7]. The differences being that, for the new breed of commodity vector accelerators above, both the hardware and the software are riding on the commodity ecosystem, whereas these older accelerators that were dedicated to particular hardware platforms were not. As a result, one could not, for example, easily "upgrade" the accelerator as the general CPU, and/or not being able to take advantage of a vast software infrastructures that were present, i.e., by the time the software would be customized to take advantage of acceleration, the hardware would be "caught up" with general-purpose CPUs such that it would effectively be rendered "obsolete". This emphasizes the importance of the acceleration being on the same performance growth curve as dictated by the Moore's law, but the only successful way of sustaining such a growth curve had been to ride on the commodity ecosystem.

2 Challenges in Commodity Vectors

So, it is our claim that commoditization of vectors are finally here, a phenomenon of such importance after numerous years of supercomputing such that it would propel HPC in the mainstream market--- over a decade of gradual commoditization of central pieces that effectively comprised supercomputers, starting from CPUs to memory systems, the operating systems and middleware, networks such as 10GbE and Infiniband, and finally onto vector acceleration. But as pointed out, albeit from rather graphics-oriented point of view in [2], various problems still remain by which we cannot claim with 100% confidence that commodity vectors are here to stay in the HPC space. In fact, we are not even settled on what would be the governing architecture, as various hardware chips differ considerably in the architectural organizations as of current, and their evolutions could even be more disparate, rather than being on the convergence paths. The associated software platforms are still nascent at best, a historical parallel to early days of older vector computing. In fact, some of the algorithms and the software artifacts could be applied straightforwardly, but those are fairly few in number. In many cases, research and development has to be conducted at various degrees of intensity to utilize the new breed of commodity vector hardware to their fullest capabilities, just as various research were needed to harness the power and the scalability of commodity clusters. In the talk I will outline some of the details, and our recent research endeavors aimed at solving the various issues, but here are some of the highlights:

1). *Combined Commodity Vector / CPU Algorithms and their Performance Models*--- Commodity vectors such as GPUs are aimed mainly at stream processing, while CPUs are more general purpose and are more latency optimized, but still have

significant stream processing power. Moreover, communication bandwidth between commodity vector components and the CPUs are currently limited by the PCI-Express bus, whereby our measurements show that even for the most current generations being far inferior compared to naïve memory system performance (over 60GBytes/s for high-end GPU memory systems compared to 5GB/s for PCI-Express x16 Gen2). As such, devising an effective methodology as to how to divide the labor between the CPUs and commodity vectors in the system needs to be investigated. This is different from the classical vector days where algorithmic vectorization would be done gradually in a piecewise fashion in the performance critical portions of the code, whereby the merger between the scalar versus the vector processing came essentially for free.

Our recent work in this context mentioned earlier[6] comes up with a methodology and an algorithmic instantiation in Linpack whereby we virtualize the ClearSpeed accelerators as CPUs (Figure 1). This is possible when the workload can be described as divisible labor in which both CPUs and vectors could compute the same subspace in the workload, albeit with different performance characteristics. The improved scheme now allows other CPUs to be consolidated into the system, allowing us to achieve over 67 Teraflops for yet another performance-improvement Top500 submission for June, 2008.

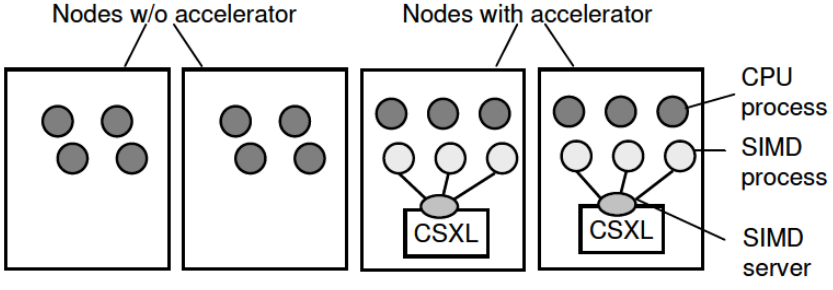


Fig. 1. Overview of Virtualization of SIMD Vector Processing in our Heterogeneous Linpack Algorithm presented in [6]

Another one of our work [11] attempts to solve such disparity in hybrid FFT algorithm where we devise a higher-performing FFT algorithm compared to the vendor Cuda FFT library (Figure 1), and moreover, construct a performance model such that the combined heterogeneous outcome in performance can be accurately described according to how we divide the labor; here the performance model was shown to be accurate to be within 5%, and effective division could be derived with a simple search for the minimal point in the performance curve in the model.

Still another work by our collaborator, Professor Takayuki Aoki's group approaches the problem in a completely different fashion---in solving the Poisson equation for a benchmark CFD problem (Himeno Benchmark), the inner loop of the algorithm principally conducts discretized stencil differentiation operations

exploiting the computational power of multiple GPUs in a compute node, whereas the CPU "driver" routine is an outer-loop coordinator that effectively synchronizes the GPUs and conducts boundary region data exchange using OpenMP. As such, the CPU stays away from the core of the inner loop. The combined parallel performance using four Nvidia 8800 GTX Ultra cards in a node essentially "blew away" the previous known result for Himeno for a single node by several factors, achieving nearly 100GigaFlops on a 3-U chassis, allowing them to win the Himeno Benchmark Contest[12] for 2008. Now Aoki's team is starting to apply the methodology to more realistic CFD and related problems.

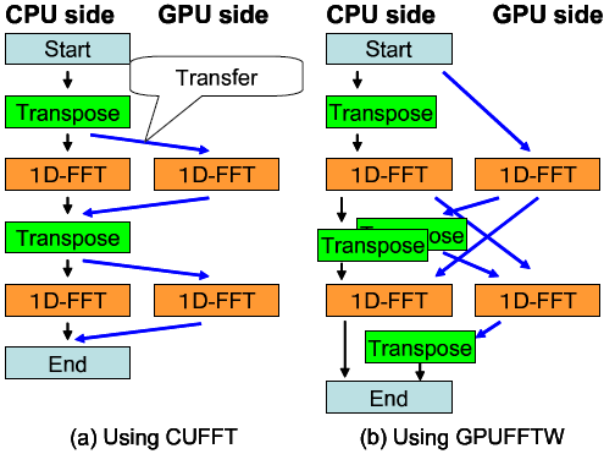


Fig. 2. Overview of our hybrid GPUFFTW Algorithm

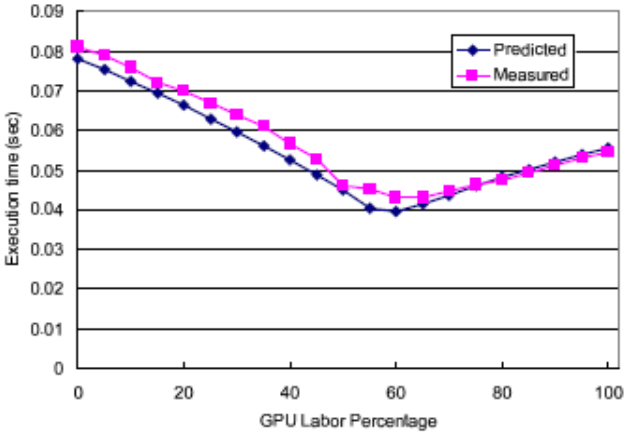


Fig. 3. Performance comparison of our hybrid 1024-point 2-D FFT algorithm—predicated performance model instantiated by a 512-point sample run versus real measurement

2). *Effective usage of memory bandwidths*---Vector accelerators, especially GPUs are heavily geared towards stream processing, and thus sport substantial memory bandwidth. Although there is some commonality to traditional vector machines, there are vast differences in how the memory system is architected, affecting how memory latency is tolerated. Traditional vector machines involve bank-interleaved memory of substantial parallelism, up to 128 on the NEC SX series[8]. Long-latency memory operations such as stride access are tolerated by such parallelism in the memory system, which would be quite expensive to implement for various reasons. GPUs, on the other hand, cannot afford such an expensive memory system. Instead, it relies heavily on multithreading to tolerate overall memory latency---a modern day GPU can have hundreds to thousands of threads with outstanding memory operations in flight. However, maximum performance as afforded by the memory system can only be achieved when threads access the memory in an orderly fashion, enabling *memory coalescing*, which is fairly large granule (64 bytes in the case of Nvidia 8800 GTX/GTS) with strict access ordering constraints. Irregular memory accesses, such as stride access, are thus particularly poor-performing on GPUs, where latency can be hidden but bandwidth is still sacrificed quite considerably. (The Cell processor is somewhat intermediate in that, they offer lower bandwidth but our measurements have found the stride access capability to be more flexible compared to GPUs.)

The issue then, is how much the traditional vector-oriented algorithm developed over the years would be applicable (or not) to modern-day commodity vectors from a performance perspective. Although there are no general answers to the question yet, there are several works in porting representative numerical kernels effectively onto commodity vectors. BLAS kernel performance on Clearspeed is particularly both fast and efficient. [9] proposes mixed usage of GPUs and CPUSs in BLAS kernels, and [10] presents a work where Level-2 BLAS achieves considerable stability over varying problem sizes on CUDA GPUs. Our recent work lead by Akira Nukada et. al. studies efficient 3-D FFT algorithm on a Nvidia CUDA GPU, where we found that some of the classical vector-based algorithms are well-applicable, but in other cases various resource constraints, particularly in the registers, plus the stringent memory coalescing requirements call for substantive adaptation of such algorithms. The net results, however, are stellar: while a 16 CPU-core AMD Opteron (2.4 Ghz) node of TSUBAME achieves less than 20 Gigaflops in double precision for a 256^3 3-D FFT calculation, a single precision version on a Nvidia 8800 GTX achieves almost 80 Gigaflops (Figure. If this were achieved in double precision, it would almost be equivalent to a single node performance on a NEC SX-8 (16 vector CPUs). It is quite interesting that, with appropriate tuning, a single graphics card almost achieves the same performance as a purpose-built high-end vector supercomputer costing 1000 times as much, and at the same semiconductor process (90nm).

Fig. 4. A 3U Compute Node with 4 Nvidia 8800GTS cards

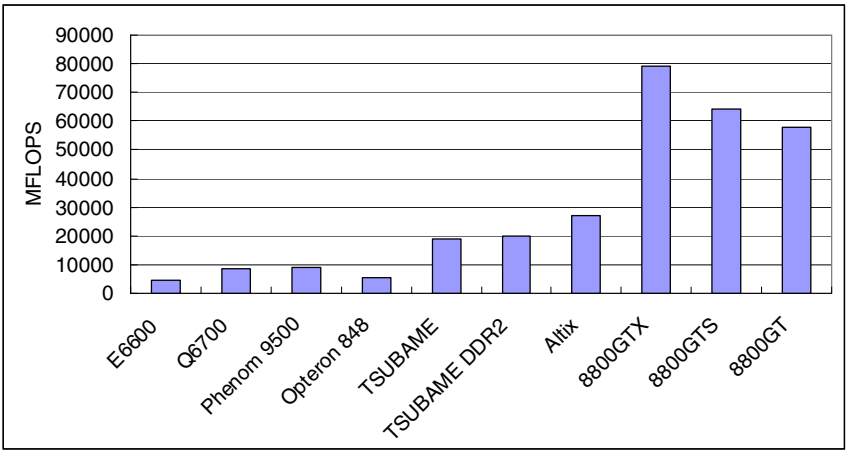


Fig. 5. Performance of our 2563 3D-FFT Library (written by Akira Nukada) in comparison to standard CPUs. Here, TSUBAME is a 16-core 2.4 Ghz Opteron node (SunFire x4600), and TSUBAME DDR2 is the 2.6 Ghz node with PC2700 DDR2 memory. For standard CPUs single and double precision performances have been found to be largely equivalent.

3) *Coping with poor vector accelerator to CPU (and network) bandwdith*---We have already mentioned that there is a vast disparity between memory bandwidths esp. of GPUs versus the achievable transfer speed on a PCI-Express bus. Although this could be alleviated somewhat by stronger integration of CPUs and vector accelerators as is the case for gaming consoles such as Sony PlayStation 3 or Microsoft Xbox 360, such disparity will likely persist at some scale due to substantial increase in point-to-point data transfer rates of directly-attached memory versus those that are not.

Given such fundamental restrictions in bandwidth, the current major workarounds are as follows 1) offload only non-bandwidth intensive application kernels to a vector accelerator 2) confine (almost) the entire application kernel to reside on the vector accelerator, or 3) attempt to hide the transfer latency by overlapping computation with data transfer. Of these, 1) is the approach used in dense problems, in which data transfer relative to computation is minimal. 2) is often used in many applications to date, including applications we have written that exploits our 3-D FFT above. In particular, we were able to realize a 3-D protein docking application whose main driver loop resides entirely on the card, and whose computation is dominated by 3-D convolution / 3-D FFT, achieving almost 50 Gigaflops on a Nvidia 8800 GTS, avoiding the major PCI-Express transfer bottleneck (Figure 6). However, this is not always applicable, and in fact could be difficult when the scalar bottleneck within the kernel becomes more dominant. 3) would be effective albeit in a smaller way, as the ability to overlap would depend on a good balance between computation and communication in the first place. Instead, the core solution to the problem from the software perspective would be to devise new algorithms such that 1)-3) or other methods would be applicable. Again, these may call for substantive research and development.

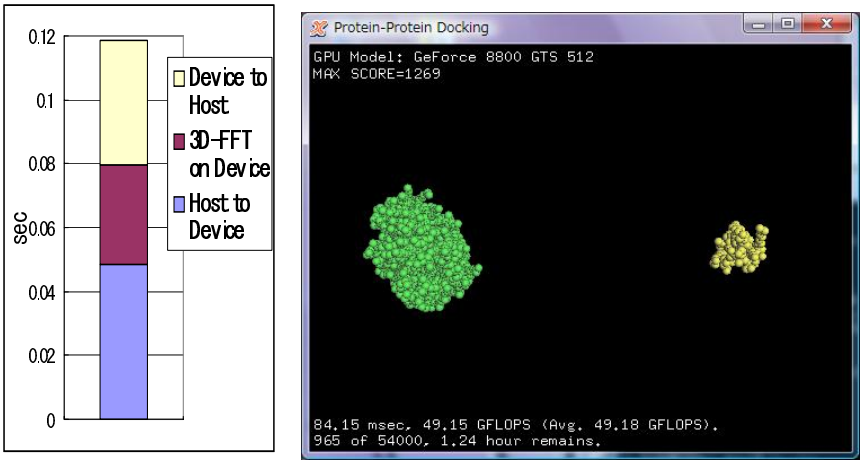


Fig. 6. The graph on the left hand sides shows the overhead of PCI-E transfer of 256^3 3-D FFT (using PCI-E Gen 2 x16). The docking application on the right eliminates the transfer overhead by combining the entire application loop, including 3-D FFT, within the GPU card. The sample run using Nvidia 8800 GTS is exhibiting almost 50 Gigaflops per card, largely preserving the 3-D FFT performance.

Other difficulties include general programming issues, ranging from the higher-level language model that would exploit the SIMD/thread parallelism while realizing orderly access to memory, as well as the actual software infrastructural support in program development, such as compilers, debuggers, various libraries, etc. Another issue is general reliability including numerical accuracy and fault tolerance. Some of the problems resemble those for the early days of cluster computing, and still would require several years of research to have them 100% resolved, despite the accumulation of knowledge and experiences from the cluster days, due to a very different computational hardware underneath. Nevertheless, as was with clusters and classical vectors, such continued research and the resulting innovation, as well as user education, will likely allow commodity vectors to be used in an easy fashion

3 The Future of Commodity Vectors

As the commodity vectors progress, would there be convergence in the architectures, just was as the case for standard CPUs to x86, due to software platform issues, or instead, diverge and become more hybrid in order to exploit the various hardware advantages that are given rise due to the nature of the target application class? Would they still be subordinates to standard CPUs, where only specific workloads are offloaded, or would they become core compute engines for most HPC workloads, demoting the standard CPUs to subordinate status for essentially running errands? What would be the key technology advances that would sustain the tremendous performance growth, not only in hardware---3-D chip design, optical chip-chip interconnect, terabyte/s-bandwidth memory systems, etc.--- but also in software, to cope with the peaky and highly parallel nature of the architecture, where the approach could be leveraging much from classical vectors (i.e., rebirth), or will we be diverging considerably, due to the commodity nature of the hardware? These and other questions are still unanswered, but their exploitations would lead to promising rise of the performance and applicability of commodity vectors, allowing us to reach not only the petaflops, but even exaflops and beyond in the very near future.

Acknowledgements

This work is supported in part by Microsoft Technical Computing Initiative, and also in part by Japan Science and Technology Agency as a CREST research program entitled ``Ultra-Low-Power HPC''. The work represents numerous efforts by our commodity vector / low power computing team, including Toshio Endo, Akira Nukada, Naoya Maruyama, Yutoh Hosogaya, Yasuhiko Ogata, and other collaborators including Profs. Takayuki Aoki, Yutaka Akiyama, and Fumikazu Konishi at Tokyo Institute of Technology.

References

[1] Meuer, H.W., Strohmaier, E., Dongarra, J.J., Simon, H.D.: The Top500 Supercomputer Sites (1993), http://www.top500.org

[2] Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., Phillips, J.: GPU Computing. In: Proceesings of the IEEE, vol. 96(5), pp. 879–899. The IEEE Press, Los Alamitos (2008)

[3] Shirts, M., Pande, V.: Screen Savers of the World Unite! The Science Magazine, the American Association for the Advancement of Science 290(2498), 1903–1904 (2000), http://folding.stanford.edu/

[4] Chen, T., Raghavan, R., Dale, J.N., Iwata, E.: Cell Broadband Engine Architecture and its first implementation—A performance view. IBM Journal of Research and Development 51(5) (August 2007)

[5] Clearspeed Whitepaper: CSX Processor Architecture, http://www.clearspeed.com/docs/resources/ClearSpeed_Architecture_Whitepaper_Feb07v2.pdf

[6] Endo, T., Matsuoka, S.: Massive Supercomputing Coping with Heterogeneity of Modern Accelerators. In: Proc. IEEE International Parallel and Distributed Processing Symposium. The IEEE CS Press, Los Alamitos (2008) (CD-ROM)

[7] Susukita, R., Ebisuzaki, T., et al.: Hardware accelerator for molecular dynamics: MDGRAPE-2. Computer Physics Communications 155(2), 115–131 (2003)

[8] To, Y., Furui, T., Nishikawa, T., Yamamoto, M., Inoue, K.: YA Development Concept of Supercomputer SX-8. NEC Technical Journal (In Japanese) 58(4), 3–6 (2005)

[9] Ohshima, S., Kise, K., Katagiri, T., Yuba, T.: Parallel processing of matrix multiplication in a cpu and gpu heterogeneous environment! In: Daydé, M., Palma, J.M.L.M., Coutinho, Á.L.G.A., Pacitti, E., Lopes, J.C. (eds.) VECPAR 2006. LNCS, vol. 4395, pp. 305–318. Springer, Heidelberg (2007)

[10] Fujimoto, N.: Faster Matrix-Vector Multiplication on GeForce 8800GTX. In: Proc. LSPP Workshop 2008, IPDPS 2008 Proceedings. The IEEE CS Press, Los Alamitos (2008) (CD-ROM)

[11] Ogata, Y., Endo, T., Maruyama, N., Matsuoka, S.: An Efficient, Model-Based CPU-GPU Heterogeneous FFT Library. In: Proc. HCW 2008: 17th International Heterogeneity in Computing Workshop, IPDPS 2008 Proceedings. The IEEE CS Press, Los Alamitos (2008) (CD-ROM)

[12] Himeno, R., et al.: The Himeno Benchmark Contest, Riken (2002), http://accc.riken.jp/HPC/HimenoBMT/contest_e.html

MOPS – A Morphodynamical Prediction System on Cluster Computers

Hartmut Kapitza

GKSS Research Centre, Institute for Coastal Research,
Max-Planck-Str. 1, 21502 Geesthacht, Germany
Ph.: +49(0)4152-87-1846; Fax: +49(0)4152-87-2818
hartmut.kapitza@gkss.de
http://www.gkss.de

Abstract. The simulation of morphodynamical processes with a 3D model requires high horizontal and vertical resolution at least in some parts of the model domain. In particular, all areas with steep bathymetry gradients like tidal rivulets or shipping channels require highly resolved simulations. Since it is not feasible to run the total model domain with the same high resolution everywhere, this problem calls for a multiply nested approach. Still, the amount of grid points necessary for a multiply nested simulation is enormous. Since in shallow areas the influence of wave action on the bottom shear stress becomes important a wave model particularly suitable for shallow water is coupled to the hydrodynamics. The integrated system is implemented on a Linux cluster using the MPI library. Performance results for different types of model coupling are presented.

Keywords: hydrodynamics, wave model, hierarchical nesting, cluster computing, morphodynamics.

Topics: Large Scale Simulations in CS&E, Parallel and Distributed Computing, Cluster Computing.

1 Introduction

In the light of a possible sea level rise due to climate change coastal engineers are interested in estimates of the impact of such a change on coastal protection works like dikes. This motivates the development of a model system integrating a current module, a wave module, and a morphodynamical module. Fig. 1 symbolizes the physical processes involved. Besides tidal motion the main driving force for currents is the stress executed by wind on the surface. This stress is also a main driving force for the generation of waves. Waves and currents also interact, and both are responsible for a stress force acting on the bottom. When the bottom is allowed to change (erosion and/or deposition) and when this change is fed back into the dynamical system the process is called morphodynamics.

This morphodynamical prediction system (MOPS) is still in development. In particular the morphodynamics part is still missing. But the most important

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 63–68, 2008.

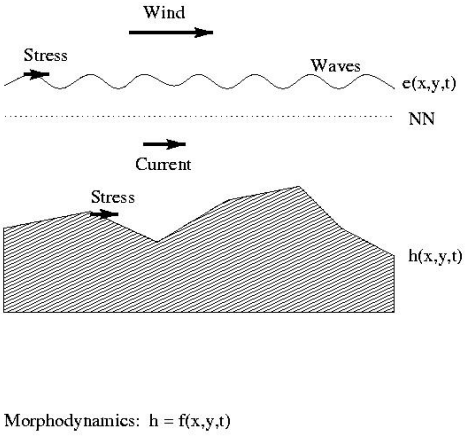


Fig. 1. Physical processes involved in a morphodynamical prediction system

prerequisites for estimating a reasonable bottom shear stress to be used for morphodynamical processes are a coupled current and wave model.

2 Component Models

The hydrodynamical model used is based on TRIM3D from Casulli and co-workers in Trento, Italy ([1]). It is a finite difference model discretized on a staggered Arakawa-C cartesian grid. Optionally it allows inclusion of baroclinic and non-hydrostatic terms, which are both not relevant for the cases presented here. We have extended the original model to allow for a focused view on the area of interest. The focus is realized by a set of hierarchical grids with increasing refinement (usually by a factor of 2), where the boundary conditions of the finer grids are provided by the results of the next coarser grid. For our test application to be described later a staggering level of 4 was used with horizontal resolutions varying from 800 to 100 m. The flow of information is still one-way from coarse to fine. A two-way nesting providing part of the unresolved coarse grid terms by fine grid results would constitute a major improvement and is on the to-do-list. A further extension of TRIM3D was its parallelization for distributed memory systems. A domain decomposition with explicit message passing using the MPI-Library was chosen.

The wave model is a spectral model especially adapted for applications in shallow waters with strong bathymetric gradients ([2]). It solves an equation for energy density taking into account wave generation by wind and non-linear dissipation effects due to wave breaking. Wave model and current model interact in two ways. On one hand water depth and current velocity influence the wave period, while on the other hand wave energy can also be transferred to currents by terms called radiation stress. This effect occurs primarily in shallow water with strong energy gradients and can lead to significant long shore currents. Certainly, the effect is strongest during strong wave periods like storms.

3 Test Case

The test bed for the combined system was the Hörnum tidal basin between the islands of Sylt, Amrum and Föhr in the German Bight. Four nested grids with horizontal resolutions of 800, 400, 200, and 100 m were used. Fig. 2 shows a satellite picture with the position of the finest grid indicated by the yellow rectangle. The coarsest grid was driven by data from the BSH (Bundesamt für Seeschifffahrt und Hydrographie). Simulating time was a period of two years from November 1999 to October 2001, which also includes a heavy storm (*Anatol* on 3rd and 4th of December 1999). Fig. 3 shows the bathymetry of the finest grid and in Fig. 4 the typical surface velocity pattern at maximum ebb tide is presented. Very strong currents in the main tidal channel can be seen as well as strong crosswise currents over the shallows at the southern tip of Sylt.

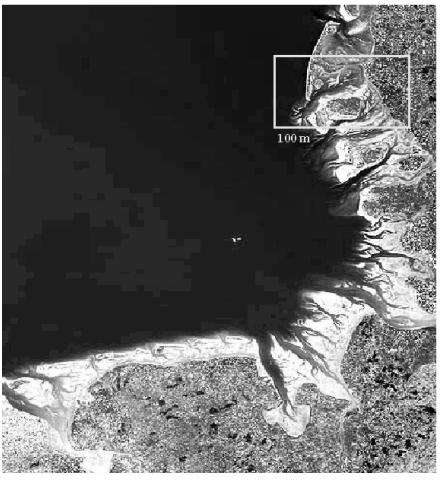


Fig. 2. German Bight with rectangle indicating the location of the finest grid

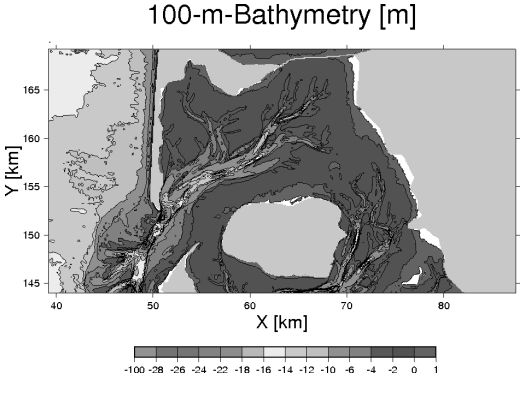


Fig. 3. Bathymetry of the 100 m resolution grid

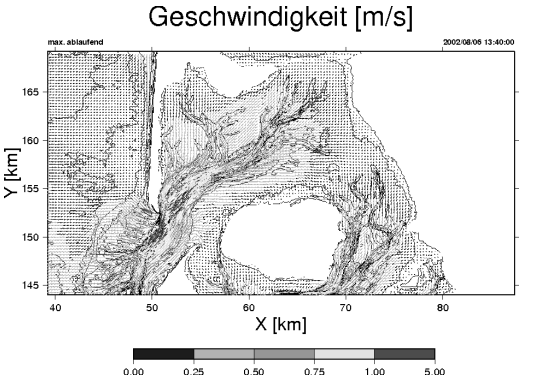


Fig. 4. Surface velocity vectors at maximum ebb tide

4 Timing Results

In a first setup for the model system the wave model was still not parallelized. Therefore, it was placed on a separate processor on the Linux cluster with a total of 64 2.4 GHz Intel Xeon processors (Fig. 5, left panel). Fig. 6 shows the timing of the model system with data exchange directions and positions in time. It turned out that the wave model was the bottle neck of the system. When run on the finest grid it took 3 times longer than real time. On the other hand the current model when run in fully nested mode on 8 processors was about 8 times faster than reality. Since the latter timing was considered to be feasible in terms of total CPU time for the whole simulation period the wave model was adapted to a much coarser 400 m resolution on the finest grid domain in order to give approximately the same CPU demands. This required a lot of interpolation back and forth, which is certainly not ideal. Nevertheless, the system was run stably and produced reasonable results giving estimates of the wave energy flux on the coast line (which is the relevant parameter for coastal engineers).

In the meantime the model system has been migrated to another Linux cluster consisting of 24 nodes with 2 dual-core 2.2 GHz Opteron processors on each node.

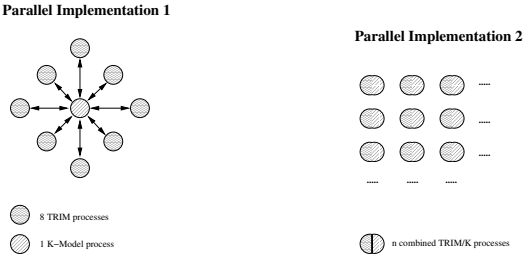


Fig. 5. Parallelization patterns. Left: Original version with sequential K-model. Right: Fully parallelized system.

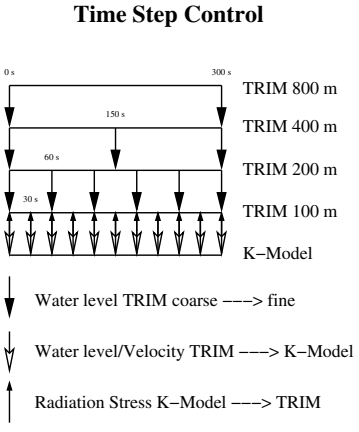


Fig. 6. Time step control of the coupled system. The arrows indicate data exchange.

NP	NPN	CPU	S	E
1	1	7312	1.00	1.00
3	1	2335	3.13	1.04
6	1	1147	6.37	1.06
12	1	572	12.78	1.06
24	1	264	27.70	1.15
24	2	323	22.64	0.94
48	2	235	31.11	0.65
96	4	153	47.79	0.50

Fig. 7. Timing results for the Opteron cluster. *NP* is the number of processors used, *NPN* denotes the number of processors per node utilized, *CPU* is total CPU time in seconds for a 24-hour simulation, S is the speedup, and E the efficiency.

Fig. 7 shows performance results for a 24-hour simulation of the hydrodynamic code alone. Interestingly the Opteron cluster shows super-linear speedup as long as only one processor per node is used. This is probably a cache effect. The more processors are used the smaller the individual sub-problems become fitting more easily into the cache. Comparing the results for 24 processors but using 2 processors per node instead of 1 shows a significant drop in efficiency. This is an indication of competition among several processors on limited resources on a single node. In particular the output of the model is organized such that each processor writes preliminary output on a scratch disk private to each node. After the model run is finished a post-processing script collects all the sub-domain files and transforms them into a single netCDF output file. The scratch disk is shared among all processors on a node which could explain the drop in performance when using more than one processor per node.

In the next stage of development the wave model was parallelized with the same domain decomposition approach as the current model (Fig. 5, right panel). The coupling can now be much tighter (data exchange every time step), and the synchronization is just a question of defining the sub-domains in order to achieve load balance. Now the wave model can also be designed in a multiply nested way, but it turned out quickly that now the computing time increases by approximately a factor of 10. Running the coupled and fully parallelized system on a much larger number of processors is not really a solution since then the sub-domains become very small increasing the communication overhead. Therefore, it is still a matter of research how to apply this system optimally on a cluster computer. One solution might be to keep on running the wave model on the coarser grids only. Another solution could be to switch on the wave model only in situations where the coupling has significant influence on the results. As it turned out from the two years simulation period only strong wind events create strong enough waves to be of importance for the currents and bottom shear stresses. Most of the time the differences between runs with or without waves were negligible.

5 Conclusions and Outlook

In conclusion it was shown that a coupled system of current and wave prediction results in a gain of quality of the results. It still needs to be shown that the inclusion of a morphodynamic sub-module benefits from these results. Furthermore, the CPU requirements of the fully nested system for both currents and waves are prohibiting for routine forecasts. There still needs to be found a way to simplify the approach without loosing too much of forecast skill.

References

1. Casulli, V., Stelling, G.S.: Numerical Simulation of 3D Quasi-Hydrostatic, Free-Surface Flows. J. Hydraulic Eng. 124, 678–686 (1998)
2. Schneggenburger, C., Günther, H., Rosenthal, W.: Spectral Wave Modelling with Non-Linear Dissipation: Validation and Application in a Coastal Tidal Environment. Coastal Eng. 41, 201–235 (2000)

Implementing a Parallel NetCDF Interface for Seamless Remote I/O Using Multi-dimensional Data

Yuichi Tsujita

Department of Electronic Engineering and Computer Science,
School of Engineering, Kinki University
1 Umenobe, Takaya, Higashi-Hiroshima, Hiroshima 739-2116, Japan
Tel.: +81-82-434-7000; Fax: +81-82-434-7011
tsujita@hiro.kindai.ac.jp

Abstract. Parallel netCDF supports parallel I/O operations for a view of data as a collection of self-describing, portable, and array-oriented objects that can be accessed through a simple interface. Its parallel I/O operations are realized with the help of an MPI-I/O library. However, such the operations are not available in remote I/O operations. So, a remote I/O mechanism of a Stampi library was introduced in an MPI layer of the parallel netCDF to realize such the operations. This system was evaluated on two interconnected PC clusters, and sufficient performance was achieved with a huge amount of data.

Corresponding topics: Parallel and Distributed Computing, Cluster Computing.

1 Introduction

Recent parallel scientific computations require not only a huge amount of computing power but also a huge amount of data storages. Scientific computations usually output intermediate data for check-point restart or analysis by using a visualization software after computation. In such the computation, common portable data format and I/O interfaces are very useful because users want to concentrate in their computations.

Several kinds of I/O interfaces such as netCDF [1] support such data format and simple I/O interface. NetCDF was developed to support a view of data as a collection of self-describing, portable, and array-oriented objects that can be accessed through a simple interface. It provides a portable I/O interface which supports not only fixed size arrays but also variable size arrays. It has been widely used in many kinds of scientific computations such as meteorology.

NetCDF is a useful interface library, however, it only supports serialized I/O operations. As a result, such I/O operations would be a bottleneck in parallel computation. A parallel I/O interface named parallel netCDF (hereafter PnetCDF) was developed in order to realize effective parallel I/O operations for

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 69–82, 2008.

netCDF data with the help of an MPI-I/O library [2] such as ROMIO [3]. It succeeded in scientific computation [4] and several visualization softwares support its data format. However, the same operations among computers have not been available. Seamless remote I/O is useful for a user's client application such as a parallelized visualization software. A remote MPI-I/O mechanism of a Stampi library [5] has been introduced in a PnetCDF's MPI layer in order to realize this mechanism. The MPI-I/O mechanism supports automatic selection of local and remote I/O operations based on a target computer name which is specified in an `MPI_Info` object by an MPI program. MPI functions of a PnetCDF library have been replaced with the Stampi's MPI functions to support seamless remote I/O operations through a PnetCDF interface without paying attention to complexity and heterogeneity in underlying communication and I/O systems. In this paper, architecture and execution mechanism of this system are discussed in Section 2. Performance results are reported in Section 3. Related work is remarked in Section 4, followed by conclusions in Section 5.

2 Remote I/O through a PnetCDF Interface

In this section, we describe decompositions of multi-dimensional data, a derived data type associated with the decompositions, and a remote I/O system with such data type.

2.1 Decompositions of Multi-dimensional Data and Associated Derived Data Types

In computer simulations, multi-dimensional data are frequently used to store calculated results, for example. In this section, decompositions of n-dimensional data sets and associated derived data types are discussed. We denote lengths of each axis with index of $1, 2, \ldots, n$ as $L_1, L_2, \ldots, L_n$, respectively. In the data sets, we suppose an array data in a C program. Let us assume that index of 1 is the most inner index and stands for the most significant dimension. On the other hand, index of n is the most outer index and stands for the least significant dimension.

Decompositions along the most inner and outer indexes make derived data types as shown in Figures 1 (a) and (b), respectively. In Fig. 1 (a), each user process accesses dotted non-contiguous data fields with L_1/np for a block length and $rank \times L_1/np$ for a stride length, where np and $rank$ stand for the number of user processes and an unique ID in an MPI program, respectively. Each offset is specified not to overwrite the data fields each other. On the other hand, Fig. 1 (b) shows a simple derived data type split evenly along the most outer index. It is clear that splitting evenly along the most significant axis provides the most complex data image and that along the least significant axis provides the most simplest one. This kind of data type is easily created by MPI functions for derived data types such as `MPI_Type_vector()`. In a PnetCDF interface, several kinds of MPI functions are used to create such derived data types. This issue is discussed in the next section.

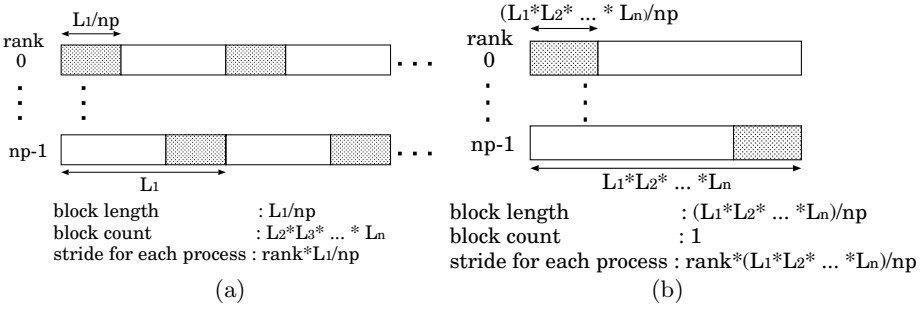


Fig. 1. Derived data types split evenly along (a) the most inner and (b) the most outer indexes for np user processes

2.2 Support of PnetCDF in Remote I/O

PnetCDF supports many kinds of parallel I/O interface functions, and their parallel I/O mechanism is realized by using MPI functions inside them. As an example, several PnetCDF functions and used MPI functions inside them are listed in Table 1. In the PnetCDF, a native MPI library such as MPICH is used. Parallel I/O operations inside the same MPI implementations are available, however, remote I/O operations are not. As Stampi supports MPI functions listed in the table, we have replaced native MPI functions with Stampi's MPI functions which start with `_MPI_` in order to develop the remote I/O system. As the functions switch to local or remote I/O operations based on a destination computer automatically, seamless I/O operations are available in the PnetCDF layer. In the local I/O, portable interface functions of a native MPI library which start with `PMPI_` are called inside the Stampi layer. While in the remote I/O, an I/O request and associated parameters such as message data size and a data type are transfered to a corresponding remote MPI-I/O process. The MPI-I/O process plays requested I/O operations. Details of this mechanism are discussed in 2.5.

Table 1. Typical PnetCDF functions and MPI functions which are called inside them

PnetCDF functions	Used MPI functions
`ncmpi_create()`, `ncmpi_open()`	`MPI_File_open()`, `MPI_File_delete()`, etc.
`ncmpi_put_var_int()`	`MPI_Comm_rank()`, `MPI_File_set_view()`, `MPI_Type_hvector()`, `MPI_Type_commit()`, `MPI_Type_free()`, `MPI_File_write()`, etc.
`ncmpi_put_vars_int_all()`	`MPI_Comm_rank()`, `MPI_File_set_view()`, `MPI_Type_hvector()`, `MPI_Type_commit()`, `MPI_Type_free()`, `MPI_File_write_all()`, etc.
`ncmpi_close()`	`MPI_Allreduce()`, `MPI_File_close()`, etc.

2.3 Architecture of a Remote I/O Mechanism

Architecture of the I/O mechanism is depicted in Figure 2. MPI communications inside a computer are carried out by using a native MPI library. When a PnetCDF interface is called for I/O operations inside a computer, associated Stampi's MPI interface functions are called, and high performance I/O operations are carried out by calling a native MPI library by the Stampi's functions. If the native one is not available, UNIX I/O functions are used instead of it.

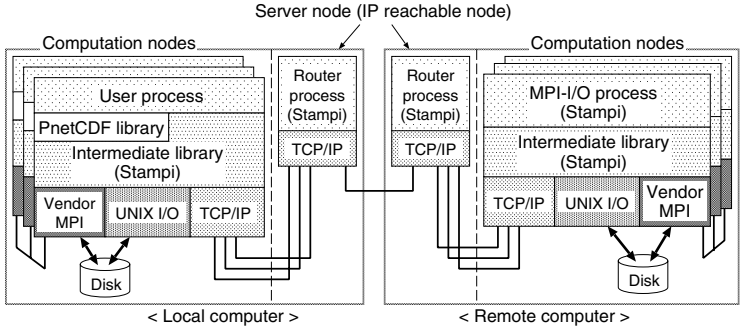


Fig. 2. Architecture of a remote I/O mechanism with PnetCDF interface support

While in MPI communications among computers, user's MPI processes invoke MPI processes on a remote computer by using rsh or ssh when a spawn function (`MPI_Comm_spawn()` or `MPI_Comm_spawn_multiple()`) is called. MPI communications between the local and remote MPI processes are carried out via inter-connections established by TCP sockets. If computation nodes of a computer can not communicate outside directly, a router process is invoked on an IP reachable node to relay message data among computers. For remote I/O operations, an MPI-I/O processes are invoked on a remote computer to play I/O operations instead of the MPI processes when an MPI function to open such as `MPI_File_open()` is called. In I/O operations by the MPI-I/O processes, a native MPI library is used via a Stampi's MPI interface library as default. If the native one does not support MPI-I/O operations, UNIX I/O functions are used instead of it.

2.4 Execution Mechanism

An execution mechanism of the remote I/O operations is illustrated in Figure 3. Firstly, a user issues a Stampi's start-up command to initiate a Stampi starter (jmpirun) (**1**). The starter process invokes a native MPI start-up process (MPI starter) such as mpirun (**3**). Later the MPI starter process invokes user's MPI processes (**4**). For remote I/O operations, parameters such as a host name of a target computer, a user ID, and a work directory are specified in an `MPI_Info` object. When `ncmpi_create()` (for creating a new file) or `ncmpi_open()` (for opening an existing file) is called by the user processes, a Stampi's `MPI_File_open()`

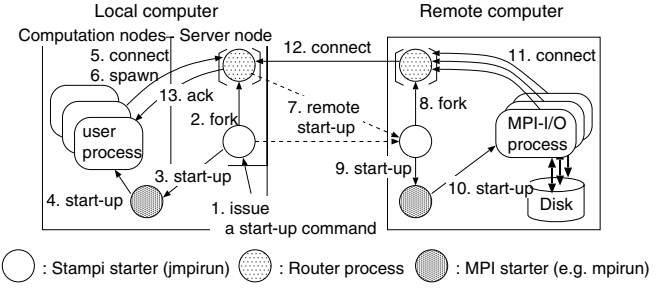


Fig. 3. Execution mechanism of remote I/O operations

is called by the PnetCDF function. After this operation, MPI-I/O processes are invoked on a remote computer according to the parameters in the `MPI_Info` object (**5**-**7**, **9**-**12**). If computation nodes where the MPI processes or MPI-I/O processes are running can not communicate outside directly, a router process is invoked on a server node by the Stampi starter (**2**, **8**). Once a communication path is established among them, remote I/O operations are available by sending I/O requests from the user processes to the MPI-I/O processes. After I/O operations, `ncmpi_close()` is called in the user processes to close a opened file, followed by calling `MPI_File_close()` to close the file and terminating the MPI-I/O processes inside it.

2.5 Execution Steps of PnetCDF Functions

I/O operations with a derived data type are essential mechanisms to support PnetCDF. Execution steps of remote I/O operations using `ncmpi_put_vars_int_all()` are explained as an example. Execution steps of I/O operations using the function are illustrated in Figure 4. Before I/O operations, we need to specify several parameters associated with the operations by using other PnetCDF and MPI functions. Inside each PnetCDF function, several MPI functions are used as listed in Table 1. Operations of this write function are grouped into two parts: one is for creation of a derived data type, and other for I/O operations by using the data type. The former part is carried out by using `MPI_Type_hvector()`, `MPI_Type_commit()`, `MPI_File_set_view()`, and so on. While the latter one is carried out by using `MPI_File_write_all()`. Execution steps for the former and the latter operations are depicted in Figures 5(a) and (b), respectively. Firstly, a derived data type is created by `MPI_Type_hvector()` and `MPI_Type_commit()` as shown in Fig. 5(a). After a file view is created by `MPI_File_set_view()`, several parameters such as a unit data type, a block length, a stride length, and so on are stored in a list-based table provided by a Stampi library in each user process. A request of the function and the parameters are transfered to a corresponding MPI-I/O process by calling a Stampi's function which starts with `_MPI_`. As the data transfer is carried out by nonblocking TCP socket connections, overlap of computation by user processes and the data transfer would be expected. After

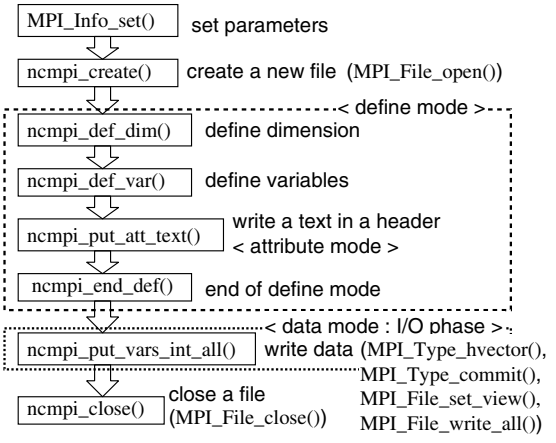
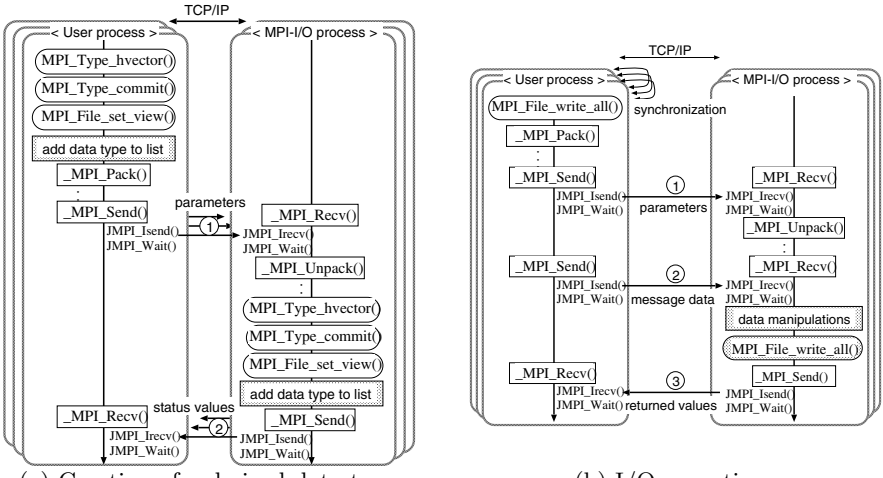


Fig. 4. Execution steps of typical collective write operations



(a) Creation of a derived data type (b) I/O operations

Fig. 5. Execution steps of (a) derived data type creation and (b) write operations. Functions in rectangles which start with `_MPI_` are Stampi's MPI interface functions.

each MPI-I/O process receives them, the same derived data type is created using the same functions, and they are stored in the similar table provided by a Stampi library in each MPI-I/O process. Each process returns a status value to a corresponding user process by using the Stampi's function at the final step.

Once we succeed to make a derived data type, we can start I/O operations using it as shown in Fig. 5(b). Associated MPI functions such as `MPI_File_write_all()` are called in an MPI layer of a PnetCDF interface. I/O requests of each MPI functions and associated parameters are transfered to a corresponding MPI-I/O process, and message data are also transfered later. Once each MPI-I/O

process receives them, it plays I/O operations by using the derived data type. When the I/O operations finish, each MPI-I/O process returns a status value to the corresponding user process.

Concerning a derived data type, this system operates rearrangement of striped data across I/O nodes by using intra-computer communications as shown in Figure 6. Each rectangle stands for an assigned memory buffer provided by this system. This mechanism has been adopted in order to reduce performance degradation by derived data type creation. Times for rearrangement of striped data by using intra-computer communications are quite shorter than those by using inter-computer communications. We assume that there are high speed intra-computer connections inside a PC cluster because each node is connected to a high speed Ethernet switch. We can utilize full bandwidth of the switch by using a vendor MPI. On the other hand, an inter-computer connection has a narrow bandwidth because we have a single link between clusters in general. We consider that it is better to reorder striped data inside a PC cluster.

3 Performance Evaluation

This system was evaluated among two PC clusters which were connected via 1 Gbps Ethernet. Specifications of the clusters are listed in Table 2. Each cluster had one server PC node and four computation PC nodes. Interconnection between the PC nodes was established via Gigabit Ethernet switches. In the both clusters, MPICH [6] (version 1.2.7p1) was available as a native MPI library. PnetCDF version 0.9.4 was used in this system. PVFS2 [7] (version 1.4.0) was available in the PC cluster II by collecting disk spaces of four computation nodes. Network connections between the clusters were established by connecting the both switches via a FreeBSD PC node which acted as a gateway.

In performance measurement, user processes were initiated on computation nodes of the cluster I, and remote I/O operations to the PVFS2 file system were carried out by invoking the same number of MPI-I/O processes on computation nodes of the cluster II. The notation, *np* also stands for the number of MPI-I/O processes in remote I/O operations. In this test, we used three-dimensional data sets. The following four different message data sets were prepared by using an integer data type (NC_INT):

- $16 \times 16 \times 16$ (16 Kbyte)
- $64 \times 64 \times 64$ (1 Mbyte)
- $128 \times 128 \times 128$ (8 Mbyte)
- $256 \times 256 \times 256$ (64 Mbyte)

Firstly, we measured times for non-collective (`ncmpi_put(get)_var_int()`) and collective operations (`ncmpi_put(get)_vars_int_all()`) as shown in Figure 7.

It is remarked that the data were split along the z-axis evenly in the collective operations. In the both operations, the collective functions outperformed the non-collective ones at each message data size, however, the times were not minimized so much with an increase in the number of user processes. This was

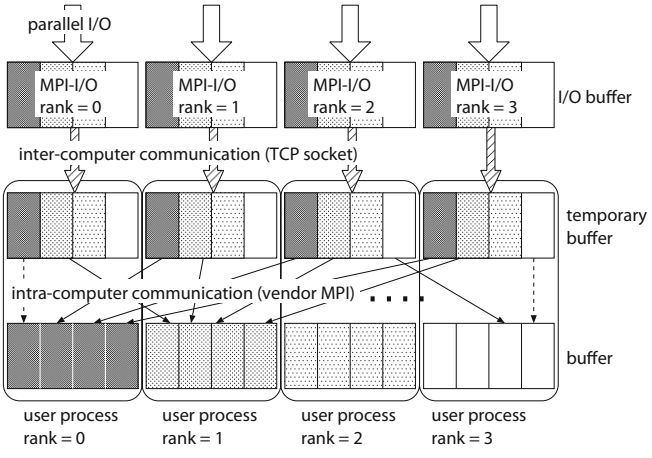
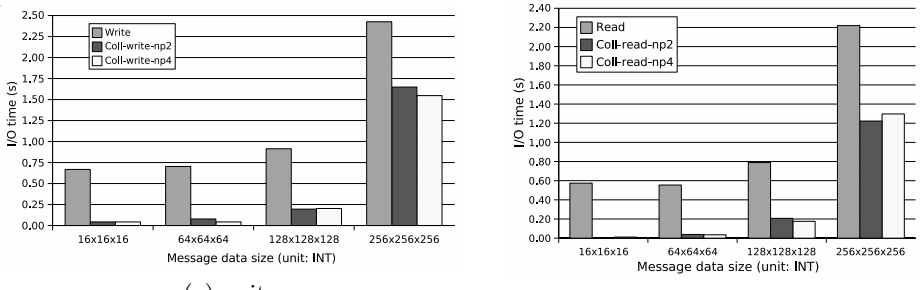


Fig. 6. Typical I/O and communications patterns in collective I/O with a derived data type



(a) write (b) read

Fig. 7. Times of non-collective and collective remote I/O operations. *Write* and *Read* denote non-collective operations. While *Coll-write* and *Coll-read* denote collective operations, where numbers which follow *np* are the number of user processes and MPI-I/O processes.

due to a bottleneck in network connection between the clusters. Inter-computer communication was a bottleneck and its operation times basically depended on total data size. Furthermore, there was a single network link between PC clusters and the total data size was constant even if we change the number of user processes. As a result, total I/O times were bounded by inter-computer network sustained bandwidth. Although we have such the restriction, this system would be usable because total I/O times are not degraded so much with an increase in the number of user processes. It is also noticed that it may be possible to increase its total performance if we have multiple physical network links by using link aggregation between PC clusters, for example.

Secondly, times for collective functions were measured with respect to axis to split a data image along. It is obviously expected that the more I/O pattern

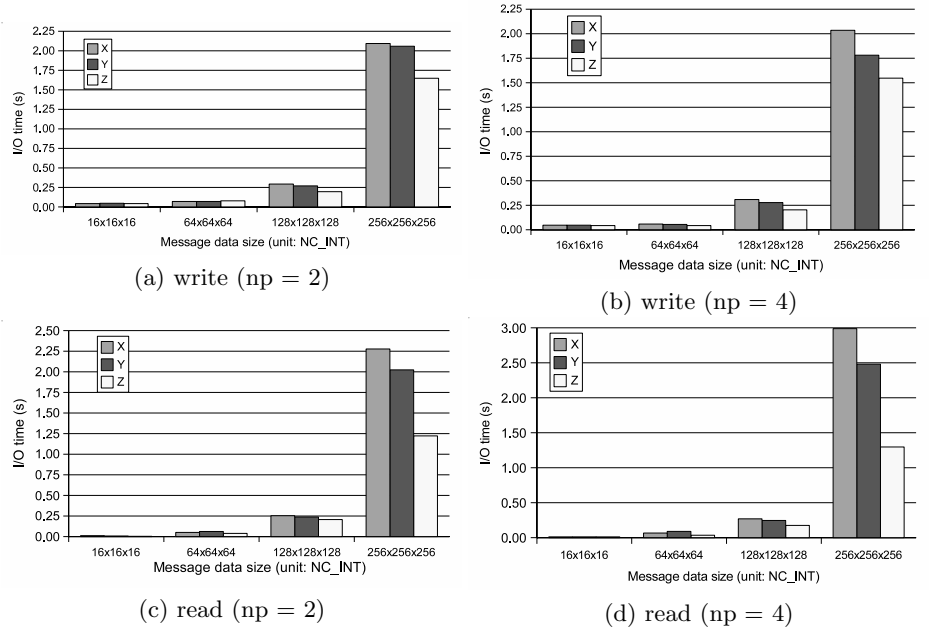
Table 2. Specifications of PC clusters used in performance measurement, where **serv** and **comp** in a bold font denote server and computation nodes, respectively

	PC cluster I	PC cluster II
serv	Dell PowerEdge800 × 1	Dell PowerEdge1600SC × 1
comp	Dell PowerEdge800 × 4	Dell PowerEdge1600SC × 4
CPU	Intel Pentium-4 3.6 GHz × 1	Intel Xeon 2.4 GHz × 2
Chipset	Intel E7221	ServerWorks GC-SL
Memory	1 Gbyte DDR2 533 SDRAM	2 Gbyte DDR 266 SDRAM
Disc system		
serv	80 Gbyte (Serial ATA) × 1	73 Gbyte (Ultra320 SCSI) × 1
comp	80 Gbyte (Serial ATA) × 1	73 Gbyte (Ultra320 SCSI) × 2
NIC	Broadcom BCM5721 (on-board PCI-Express)	Intel PRO/1000-XT (PCI-X board)
Switch	3Com SuperStack3 Switch 3812	3Com SuperStack3 Switch 4900
OS	Fedora Core 3	
kernel	2.6.12-1.1381_FC3smp(**serv**)	2.6.12-1.1381_FC3smp (**serv**)
	2.6.11-1SCOREsmp (**comp**)	2.6.11 (**comp**)
Ethernet	Broadcom tg3 v3.71b (**serv**)	Intel e1000 version 6.0.54 (**serv**)
driver	Broadcom tg3 v3.58b (**comp**)	Intel e1000 version 5.6.10.1 (**comp**)
MPI library	MPICH version 1.2.7p1	

becomes complex, the more the I/O times increase. In this test, we measured the times for splitting along x, y, and z-axes. The results are shown in Figure 8. In both the read and the write operations, splitting the data image along the z-axis provided the most shortest times, and the times for the x-axis were the worst. Difference between the times for the x and z axes was around 0.5 s. It is also noticed that the I/O times were almost the same between the cases for two and four processes except the case of $256 \times 256 \times 256$ message data in the read operations.

To find reasons for the increase in the I/O times with respect to axis to split along, we measured times for creation of a derived data type, synchronization of collective operations by `MPI_File_sync()`, and `MPI_File_write_all()` in remote I/O by using a pure MPI program. We supposed that those functions simulated I/O patterns which were carried out in the PnetCDF program.

Concerning creation of a derived data type, processing times were constant and negligible (around 80 ms for two and four user processes) in the total I/O times. Measured times of `MPI_File_sync()` in remote I/O are shown in Figure 9. They became long with an increase in the message data size because the Stampi's `MPI_File_sync()` for remote I/O synchronized all the MPI-I/O processes after I/O operations. Figure 10 shows measured I/O times of `MPI_File_write_all()` and `MPI_File_read_all()` in remote I/O. For example, there is about 0.5 s difference between the times for splitting the data image along the z-axis and others. As it is hard to simulate operations of PnetCDF functions, this might be rough analysis, however, it is concluded that the differences in the I/O times with respect to axis to split along were mainly due to an increase in I/O times



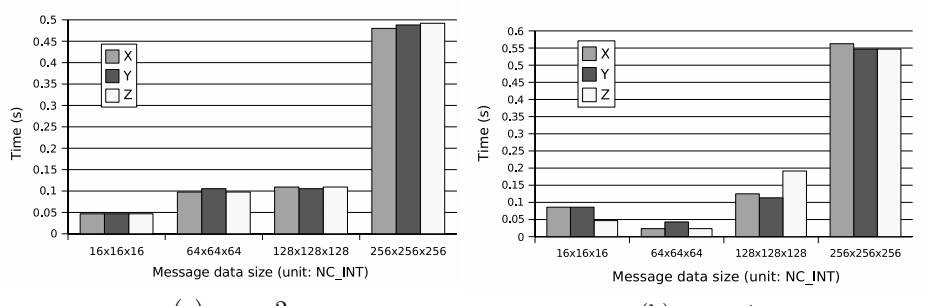
(a) write (np = 2)

(b) write (np = 4)

(c) read (np = 2)

(d) read (np = 4)

Fig. 8. Times of collective PnetCDF functions in remote I/O with respect to axis to split a three-dimensional data along. *X*, *Y*, and *Z* denote axes to split along.



(a) np = 2

(b) np = 4

Fig. 9. Times of `MPI_File_sync()` in remote I/O with (a) two and (b) four user processes and MPI-I/O processes. *X*, *Y*, and *Z* denote axes to split a three-dimensional data along.

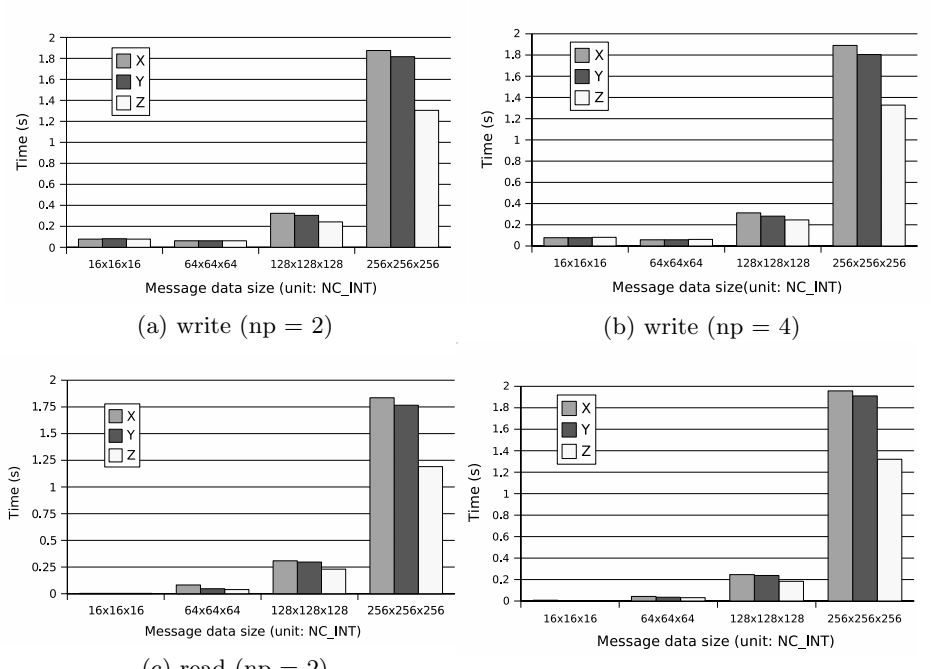
of `MPI_File_write_all()` and `MPI_File_read_all()`. To check whether this is coming from network data transfer or I/O operations on a remote computer, we also measured local I/O times on the PVFS2 file system as shown in Figure 11. It is obvious that the increase was coming from an increase in the times of the local I/O operations. Moreover, the I/O times were almost the same and did not scale

with regard to the number of user processes. This might be due to a bottleneck in intra-computer data transfer for collective operations or I/O operations on a PVFS2 file system.

4 Related Work

Providing a common data format makes data I/O operations portable for application programmers. This kind of implementations such as netCDF [8] and HDF5 [9] has been proposed. NetCDF provides a self-describing and common multi-dimensional data format and a simple interface. Its parallel I/O operations have been realized in PnetCDF, which is an extension of the interface, by introducing MPI-I/O functions as an underlying parallel I/O library [4]. On the other hand, HDF5 provides hierarchical data format in order to access huge amount of data effectively. An HDF5 interface has two objects, one is "Dataset" and another "Group". The Dataset manages multi-dimensional array data, while the Group provides relational mechanisms among objects. Parallel I/O operations are also available with this interface by introducing MPI-I/O functions as an underlying parallel I/O interface library [10].

An MPI-I/O interface in the MPI-2 standard [2] realizes parallel I/O operations in an MPI program. Several implementations of it are available such as



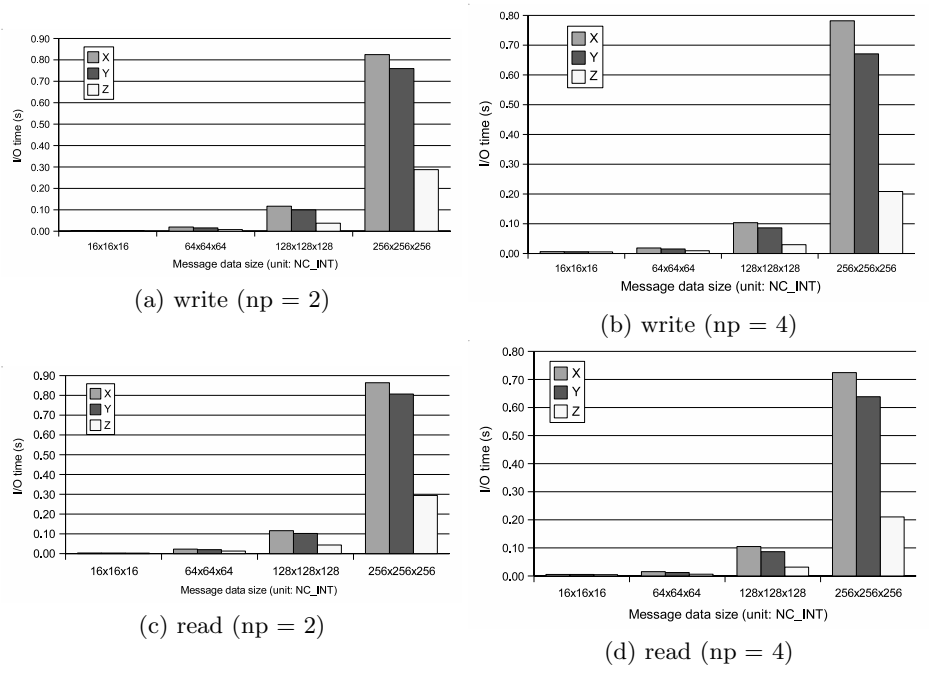
(a) write (np = 2)

(b) write (np = 4)

(c) read (np = 2)

(d) read (np = 4)

Fig. 10. Times of `MPI_File_write_all()`/ `MPI_File_read_all()` in remote I/O. *X*, *Y*, and *Z* denote axes to split a three-dimensional data along.



(a) write (np = 2)

(b) write (np = 4)

(c) read (np = 2)

(d) read (np = 4)

Fig. 11. Times of local collective I/O on a PVFS2 file system, where *X*, *Y*, and *Z* denote axes to split a three-dimensional data along

ROMIO [3]. Its MPI-I/O operations to many kinds of file systems are realized through an ADIO interface [11]. It hides heterogeneity in architectures of each systems and provides a common interface to an upper MPI-I/O layer. Remote I/O operations using the ROMIO are available with the help of RFS [12]. An RFS request handler on a remote computer receives I/O requests from client processes and calls an appropriate ADIO library. On the other hand, Stampi itself is not an MPI implementation but a bridging library among different MPI implementations. It realizes seamless MPI operations among them by using TCP socket communications.

Inter-operability among different MPI implementations is also important issue, typically in a Grid computing environment. One of the representative works is PACX-MPI [13]. It realizes inter-operable MPI communications by deploying a common MPI interface library on top of each MPI implementation in order to realize seamless MPI operations. This system provides high performance MPI communications inside the same MPI implementations and seamless operations among different ones. GridMPI [14] also realizes such mechanism in a Grid computing environment. Inter-operable MPI communications are realized through an IMPI interface by using P2P data communications. It is supported on many kinds of MPI implementations. On the other hand, Stampi realizes the similar communication mechanism with PACX-MPI with regard to a method to implement a common communication layer on top of each MPI implementation.

However, it realizes a flexible communication mechanism where invocation of proxy process is dynamically selected according to communication topology of each parallel computer.

5 Conclusion

We have developed a seamless remote I/O system using a PnetCDF interface among different MPI implementations by using a Stampi library. Its collective I/O interface outperformed its non-collective one in remote I/O. We also measured I/O times of collective one with respect to axis to split evenly along using three-dimensional data sets. Derived data types were created based on the associated splitting pattern. It was expected that the more complex the data type became, the more its I/O time increased due to an increase in the number of intra-computer data transfers for collective I/O. We have evaluated such the I/O patterns in both remote and local I/O operations. In remote I/O by using a PnetCDF interface, splitting along the most inner index provided the most complex derived data type. As a result, its I/O times were the most longest. On the other hand, times for splitting along the most outer index were the most shortest. The increase in the times with regard to axis to split along was coming from local I/O operations by using a native MPI library. The system did not scale due to a bottleneck in inter-computer data transfer or I/O operations on a PVFS2 file system, however, its performance was almost the same with respect to the number of user processes. Optimization in message data transfer between computers is considered as a future work. Implementation of non-blocking I/O functions is also considered to minimize visible I/O times.

Acknowledgment

The author would like to thank staff at Center for Computational Science and e-Science (CCSE), Japan Atomic Energy Agency (JAEA), for providing a Stampi library and giving useful information.

This research was partially supported by the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Grant-in-Aid for Young Scientists (B), 18700074 and by the CASIO Science Promotion Foundation.

References

1. Rew, R.K., Davis, G.P.: The unidata netCDF: Software for scientific data access. In: Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology, pp. 33–40. American Meteorology Society (February 1990)
2. Message Passing Interface Forum: MPI-2: Extensions to the Message-Passing Interface (July 1997)
3. Thakur, R., Gropp, W., Lusk, E.: On implementing MPI-IO portably and with high performance. In: Proceedings of the Sixth Workshop on Input/Output in Parallel and Distributed Systems, pp. 23–32 (1999)

4. Li, J., Liao, W.K., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., Zingale, M.: Parallel netCDF: A high-performance scientific I/O interface. In: SC 2003: Proceedings of the 2003 ACM/IEEE Conference on Supercomputing, p. 39. IEEE Computer Society, Los Alamitos (2003)
5. Tsujita, Y., Imamura, T., Takemiya, H., Yamagishi, N.: Stampi-I/O: A flexible parallel-I/O library for heterogeneous computing environment. In: Kranzlmüller, D., Kacsuk, P., Dongarra, J., Volkert, J. (eds.) PVM/MPI 2002. LNCS, vol. 2474, pp. 288–295. Springer, Heidelberg (2002)
6. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI Message-Passing Interface standard. Parallel Computing 22(6), 789–828 (1996)
7. PVFS2: `http://www.pvfs.org/pvfs2/`
8. Rew, R., Davis, G., Emmerson, S., Davies, H., Hartnett, E.: NetCDF User's Guide. Unidata Program Center (June 2006), `http://www.unidata.ucar.edu/software/netcdf/docs/netcdf/`
9. The National Center for Supercomputing Applications: `http://hdf.ncsa.uiuc.edu/HDF5/`
10. Ross, R., Nurmi, D., Cheng, A., Zingale, M.: A case study in application I/O on Linux clusters. In: SC 2001: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing, p. 11. ACM Press, New York (2001) (CDROM)
11. Thakur, R., Gropp, W., Lusk, E.: An abstract-device interface for implementing portable parallel-I/O interfaces. In: Proceedings of the Sixth Symposium on the Frontiers of Massively Parallel Computation, pp. 180–187 (1996)
12. Lee, J., Ma, X., Ross, R., Thakur, R., Winslett, M.: RFS: Efficient and flexible remote file access for MPI-IO. In: Proceedings of the 6th IEEE International Conference on Cluster Computing (CLUSTER 2004), pp. 71–81. IEEE Computer Society, Los Alamitos (2004)
13. Gabriel, E., Resch, M., Beisel, T., Keller, R.: Distributed computing in a heterogeneous computing environment. In: Alexandrov, V.N., Dongarra, J. (eds.) PVM/MPI 1998. LNCS, vol. 1497, pp. 180–187. Springer, Heidelberg (1998)
14. GridMPI: `http://www.gridmpi.org/`

Vectorized AES Core for High-throughput Secure Environments

Miquel Pericàs[1,2], Ricardo Chaves[4], Georgi N. Gaydadjiev[3], Stamatis Vassiliadis[3], and Mateo Valero[1,2]

[1] Computer Sciences, Barcelona Supercomputing Center
[2] Computer Architecture Department, Technical University of Catalonia
Jordi Girona, 1-3, Mòdul D6 Campus Nord, 08034 Barcelona, Spain
Tel.: +34 934 017 001; Fax: +34 934 017 055
mpericas@ac.upc.edu, mateo@ac.upc.edu
[3] Computer Engineering, Technical University of Delft
Mekelweg 4, 2628 CD Delft, The Netherlands
Tel.: +31 15 2786196; Fax: +31 15 2784898
g.n.gaydadjiev@tudelft.nl, s.vassiliadis@tudeflt.nl
[4] Instituto Superior Tecnico, INESC-ID
ricardo.chaves@inesc-id.pt

Abstract. Parallelism has long been used to increase the throughput of applications that process independent data. With the advent of multicore technology designers and programmers are increasingly forced to think in parallel. In this paper we present the evaluation of an encryption core capable of handling multiple data streams. The design is oriented towards future scenarios for internet, where throughput capacity requirements together with privacy and integrity will be critical for both personal and corporate users. To power such scenarios we present a technique that increases the efficiency of memory bandwidth utilization of cryptographic cores. We propose to feed cryptographic engines with multiple streams to better exploit the available bandwidth. To validate our claims, we have developed an AES core capable of encrypting two streams in parallel using either ECB or CBC modes. Our AES core implementation consumes trivial amount of resources when a Virtex-II Pro FPGA device is targeted.

Keywords: Parallel and Distributed Computing, Encryption.

1 Introduction

Advances in semiconductor technology have enabled industry to manufacture cores with hundreds of millions of transistors. Industry is exploiting this feature to implement chip-level parallelism in the form of multi-core on chip architectures. While 2-8 multicore chips are now common in the market it is expected that this trend will continue with even larger amounts of cores. Programmers and designers will find themselves forced into thinking concurrently in order to efficiently exploit such platforms.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 83–94, 2008.

Parallelism is, of course, not a new concept and has been implemented extensively in the past. Since the earliest machines this technique has been used to improve throughput. Parallelism can be found on all levels, from the smallest circuits to parallel clusters. From the programmer point of view, there are several ways in which to express parallel programs. One class are concurrent programming models. They map directly onto multicore architectures, but have the disadvantage that they leave the parallelization to the programmer, a task which has been shown to be often quite complex. A different way to exploit parallelism is by using SIMD programming techniques. Vector processors, for example, operate on entire vectors instead of scalar types. This programming model is effective and simple, as it retains the sequential property of single-threaded programs. However, it requires data parallelism with strict organizations in memory.

In this paper we investigate how parallelization can be used to achieve high data transfer performance in future high-throughput networks. Personal users and companies are placing growing demands of security on devices they use for their daily work. Four requirements are in demand: privacy, authentication, replay protection and message integrity. For this reason, implementations of Virtual Private Networks (VPN) rely more and more on technologies such as IPsec to secure the communication links.

A technology such as IPsec can work in two modes. In *transport mode*, the endpoint computers perform the security processing. In *tunnel mode*, packet traffic is secured by a single node for the entire computer network. In case of large networks, high performance encryption devices are required. Such is also the case with *mobile* VPNs. In a mobile VPN a device such as a handheld can have secure access to a corporate LAN to securely perform such tasks as reading email or using remote terminal sessions. It is expected that this type of networks will grow very fast in popularity in the near future.

One of the most important encryption algorithms supported by many different protocols is the Advanced Encryption Standard (AES). This encryption algorithm encrypts/decrypts blocks of 128 bits of data in 10, 12 or 14 serial stages, using 128, 192 or 256 bit-keys, respectively. To simplify our study, but without loss of generality, we will be focusing only on AES using a key size of 128 bits. In this variant the algorithm performs 10 stages to encrypt/decrypt one data block.

There are several ways in which a stream of data can be encrypted. These are referred to as *Block Cypher* modes of operation. Most of these modes require an *Initialization Vector*, which is a fixed block of data used to trigger the encryption mode. The simplest mode is the *Electronic CodeBook* mode (ECB). In this mode the data stream is partitioned into blocks of equal length and all the resulting blocks are encrypted independently. The obvious benefit of this scheme is high parallelism. All blocks that make up the stream can be encrypted simultaneously. The disadvantage of this scheme are known security concerns. More precisely, ECB does not provide good confidentiality as it does not hide data patterns well. To come up with a more robust solution several modes have been introduced. The most common of these is the *Cypher Block Chaining* mode (CBC). In this mode,

when a block is going to be encrypted, it has first to be exclusively `OR`'ed with the encryption resulting from the previous block. The first block itself is `XOR`'ed with the Initialization Vector. One drawback of this scheme is the dependency between data blocks that the cypher mode introduces. This results in a reduced efficiency concerning the available bandwidth. In this mode, the AES encryption engine can only output a block of data every 10 stages. This means that only 10% of the output capacity is used. Note, however, that the interconnect itself is independent from the engine capacity and may limit the throughput.

In cases where the available network bandwidth is larger than the single-stream CBC output, we may want to search for ways to exploit the additional bandwidth. In the domain of VPN tunnels, where a gateway is in charge of encrypting large quantities of data, we can profit from the fact that multiple (independent) channels are simultaneously active to improve the throughput of the encryption.

In this paper we propose to design AES cores capable of encrypting multiple streams at once. Using multiple streams enables parallelism and allows to better exploit the available network bandwidth. This is analogous to using vector processors to better exploit memory hierarchy in supercomputers. Further, we propose to use these cores to provide high performance file transfer between computers in the case where a large file or multiple files are being transferred. In this scenario, a user using a `scp` protocol to transfer the files, would experience a large speed-up using our proposal together with a small modification of the scp application. Finally, we also perform a pencil and paper evaluation of how the proposed core can be fitted into current system architectures.

This paper makes the following contributions:

- We observe that encrypting several streams in parallel is a way to accelerate the otherwise sequential CBC encryption.
- We implement a cryptographical unit capable of encrypting two streams in parallel using the AES encryption algorithm.
- We analyze several applications of this scheme. In particular, we discuss how this scheme can accelerate VPN networks and secure transfers of large files.
- We study two important issues relating to the implementation of the multiple-stream encryption scheme: the programming model and the system architecture.

This paper is organized as follows. Section 2 presents an overview of related work. The design of the multiple-stream encryption unit is presented in section 3 while section 4 evaluates it. Section 5 analyzes the system architecture. Section 6 discusses several issues related to this design: sections 6.1 and 6.2 analyze possible applications of this work and section 6.3 analyzes the programming model. Finally, section 7 concludes this discussion.

2 Related Work

Vectorization has long been an important technique to increase performance. Vector processors handle complete vectors instead of registers as the basic type.

Because no dependencies need to be tracked among the elements of a vector and because the memory system can be optimized to efficiently cater the large data amounts to the system, very high performance pipelines can be built. Vector implementations have been exploited mostly by numerical codes and scientific computing. These codes often feature large parallel loops that are well suited to be implemented on a vector processor.

However, attempts to build parallel implementations of cryptographic engines have not been very successful in the past, particularly those attempting to exploit algorithm-level parallelism. This can be explained intuitively. In order to provide a strong and hard-to-break encryption, algorithms rely on operations manipulating the whole data set and imposing tight dependencies among all data. Parallel execution would be possible in a higher level by encrypting multiple blocks in parallel, but this is in general precluded by the usage of block cypher modes such as CBC. Therefore, the few successful attempts to have parallel hardware accelerate an encryption procedure have relied on exploiting parallelism within the individual operations of the algorithm.

There are some examples of this kind of optimization in the literature. For example, Page et al. [1] used the SSE2 extensions of the Pentium4 are used to accelerate long precision modular multiplication. Similarly, Crandall et al. [2] used the AltiVec extension to implement long precision multiplications for the RSA algorithm. These two approaches are targeted at accelerating the 1024-bit multiplications frequently appearing in cryptographic algorithms. Another use of AltiVec is the approach introduced by Bhaskar et al. to accelerate Galois Field arithmetics [3]. This has been used to accelerate the AES algorithm, achieving an encryption rate of one block every 162 cycles. While this is impressive, it is far from what can be obtained with a hardware implementation like the one discussed in this paper. One final attempt at vectorization is the one proposed by Dixon et al., where a parallel approach is used to factorize large integers for Elliptic Curve Cryptography [4].

In this paper, instead of optimizing the basic operations, we propose a vector implementation for AES that exploits parallelism at the data level by processing multiple streams concurrently. The proposal is based on the MOLEN polymorphic architecture and programming paradigm [5,6] proposed by Vassiliadis et al. as a way to expose hardware resources to software system designers and allow them to modify and extend the processor functionality at will. The outcome of this paper is a cryptographic engine that exploits multiple streams using the vector engine paradigm. The core of the cryptographic unit is based on work by Chaves et al. within the context of the MOLEN polymorphic processor [7].

3 Multiple-Stream AES Core

To validate our assumptions we implemented an AES core capable of processing two streams concurrently. The AES-MultipleStream core (AES-MS) was implemented using the MOLEN prototype framework [8,5] and as such considers a 64-bit wide IO bus running at 100MHz. Although the width of the IO BUS has

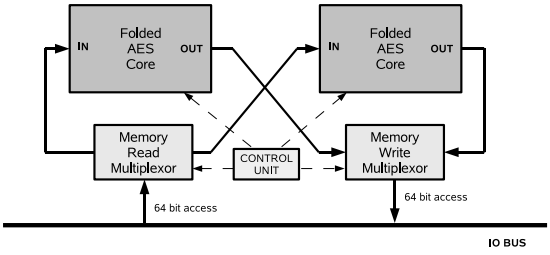


Fig. 1. Architecture of AES core handling two streams

been set at 64 bits, this is not a constraint and can be adjusted to accommodate more or less data streams. The global design of the AES core with two streams can be seen in Figure 1. It consists of two independent AES cores controlled by a Control Unit. The unit activates the AES cores when needed and manages the multiplexors that control bus access.

Each core implements an independent AES folded structure [7]. On the 64-bit/100MHz bus, a single AES core takes two cycles to read 128 bits of input data and two more cycles to output 128 bits of encrypted data. The processing amounts to 10 cycles. Thus, once the core is running it moves 256 bits every 10 cycles.

Given that in 4 out of the 10 computational AES cycles the IO Bus is used to read or write data blocks, the multiple stream version has been implemented using two streams. This results in a bus occupation of 8 out of 10 cycles (80%). No more streams can be added without changing the AES pipeline depth. The folded AES cores themselves have no information on the number of active streams; this information is handled by a small external control unit that drives the AES cores and activates the necessary multiplexors to access the external memory system.

Assuming that the AES core and the IO bus run at the same frequency, it is possible to accommodate a higher or lower number of streams depending on the IO bus width. If the bus is 128 bits wide, a 128-bit data packet can be read in a single cycle and written in another one. Given that an encryption takes 10 cycles, this would allow to encrypt up to 5 streams in parallel. Generically, with the bus and engine running at the same speed, the number of streams that can be accommodated as a function of the bus width is expressed by:

$$MaxNStreams = \lfloor 5 \cdot \frac{BusWidth}{128bits} \rfloor. \quad (1)$$

4 Performance and Results

The complete design of the two-stream AES unit was implemented in VHDL targeting the Virtex-II Pro xc2vp30-7fg676 device. Synthesis and Place & Route were both performed using Xilinx ISE 8.1. The AES core for single stream [7] spans 12 BlockRAMs and 1083 logic slices. The two-streams AES core spans

Table 1. 2-stream vs single-stream AES performance comparison

Architecture	AES – 1 stream [7]	AES – 2 streams
Cipher	Enc./Dec.	Enc./Dec.
Device	XC2VP30	XC2VP30
Number of Slices	1083	2162
Number of BRAM	12	24
Operating Frequency	100 MHz	100 MHz
Latency (cycles)	10	10
Throughput (Mbps)	1280	2560
Throughput/Slice (Mbps/s)	1.1819	1.1841

24 BlockRAMs and 2162 logic slices. The two-streams implementation puts two single-stream AES cores side-by-side and adds multiplexors that arbitrate the memory access. Some logic is shared and in the end the number of logic slices approximately doubles. Place & Route results show that the design can run at 100MHz which is the target frequency of the current MOLEN prototype. The two-streams AES-MS core consumes 17% of available BlockRAMs, 15% of all logic slices, and 38% of external IOBs while reaching a throughput of 2.56 Gbps. Table 1 shows a summary of the results and compares them against [7]. Note that the numbers provided for the original implementation of the AES core differ from those provided in [7]. This is due to some different parameters that have been used in the synthesis environment.

It should be noted that the initialization of the AES core, which includes the transmission of the key, the transmission of the initialization vector and the processor↔co-processor communication overhead, has a cost on performance. If only one data block is ciphered, the cost of initializing the AES core is an order of magnitude higher than the cost of processing the data itself. When the input data is sufficiently large, the initialization cost becomes negligible. The ciphering throughput varies from 60 Mbps for a single data block packet (128 bits) to 1.28 Gbps for a 16 kbyte packet.

5 Analysis of System Architecture

We will now present an evaluation of different system environments in which the AES core may be implemented together with performance estimations. The following cases of IO communication have been considered: the current MOLEN prototype, HyperTransport eXpansion (HTX), PCI-X, and PCI-Express (PCIe).

As mentioned earlier, the AES core has been developed and tested within the MOLEN environment. In this platform, the two-stream AES core runs at 100MHz and can encrypt and decrypt at a rate of 2.56Gbps. Considering input and output this amounts to a total traffic of 5.12Gbps, which corresponds to 80% of the total memory bandwidth in this scenario. In the following study we will

assume an AES-MS core running at 100MHz, even though the busses themselves are operated at different frequencies. We assume some sort of hardware performs the interfacing without loss of capacity.

Recently, a protocol that has emerged with good support for reconfigurable devices as coprocessors is the *point-to-point* HyperTransport protocol [9,10]. HyperTransport defines an extension protocol for coprocessors called the HyperTransport eXpansion (HTX). In the current incarnation, this standard defines a protocol that is 16 bits wide and runs at 800MHz. The bandwidth provided by a single link in single-data rate (SDR) is thus 12.8Gbps. Using two links at double-data rate (DDR) yields the maximum aggregate bandwidth of 51.8 Gbps. The single link SDR bandwidth is exactly twice that which is available in the current MOLEN prototype. Without changing the frequency of the AES core (100MHz) one could double the amount of streams (4 streams, 10.24Gbps). The remaining 2.56 Gbps are exactly the bandwidth required for one additional stream so it is possible to add a 5th stream and thus run a 5-stream AES-MS core attached to a HTX interface. Using the two HTX links with DDR would enable to accommodate up to 20 streams. Note that in this analysis the AES core and the bus operate at different frequencies. Thus we must calculate the number of streams based on available bandwidth rather than using the formula presented in section 3.

PCI-X [11] is a popular multidrop bus interconnect standard. PCI-X 1.0 features a maximum bandwidth of 8.48 Gbps at speed grade PCI-X 133, which would allow up to three streams using the AES-MS engine. A newer revision of this standard, called PCI-X 2.0, has a maximum speed grade of PCI-X 533 resulting in a bandwidth of 34.4 Gbps. This can accommodate up to 13 streams in parallel.

PCI Express (PCIe) [12] is yet another bus designed to subsitute the ancient PCI bus. Like HTX, it is a point-to-point bus, but designed to manage a wider range of devices. As a downside, it operates with slightly larger latencies. At 64 Gbps capacity (using 16 links) PCIe 1.0 would allow to interleave up to 25 streams. PCIe 2.0 runs twice as fast and would be able to accommodate up to 50 streams at maximum throughput.

Table 2. Maximum Number of Streams using 100MHz AES-MS cores

Interconnect Type	Max Bandwidth	Max Number of Streams
MOLEN Prototype	6.4 Gbps	2
HTX @ 1 Link (SDR)	12.8 Gbps	5
HTX @ 2 Links (DDR)	51.2 Gbps	20
PCI-X 133 (v1.0)	8.48 Gbps	3
PCI-X 533 (v2.0)	34.4 Gbps	13
PCIe 1.0	64 Gbps	25
PCIe 2.0	128 Gbps	50

All these numbers may seem quite high. However, if the network capacity is not as large, the AES-MS output capacity will be underutilized. In addition, as already pointed out at the end of section 4, if keys are not static and the amount of data is not sufficiently large, throughputs of Gbit/s cannot be reached as the encryption processes will be limited by the initialization phase. The previous results are summarized in Table 2. Note that in this table, *Max Bandwidth* refers to the maximum bandwidth of the interconnect, not the maximum bandwidth of the multiple stream encryption unit. Although we have not mentioned access latencies for these technologies, we assume that in stationary mode the effects of these latencies are negligible.

6 Discussion

In this section we discuss various issues related to AES-MS. So far we have implemented a core capable of exploiting multiple streams. We will now present some scenarios that can profit from the implementation and a programming model to exploit the multiple-streams feature.

6.1 Virtual Private Networks

Figure 2 (a) shows the typical architecture for a virtual private network (VPN) using unreliable connections, e.g. Internet. Such an architecture is used to securely connect multiple networks. Locally, the networks can be considered secure since the infrastructure belongs to the companies/institutions. However, on the public infrastructure no such assumptions can be made. Privacy and authentication support are required. To this end, encrypted tunnels are established. The tunnels are authenticated when a session is established. Once established, the session is kept mostly unmodified and the same keys are used to encrypt all packets.

Figure 2 (b) shows the same scenario in the case of a mobile VPN. In a mobile VPN the connection is not *network-to-network* but *client-to-network*. Every client needs to have security software installed (e.g. its own IPsec stack). The corporate side, however, looks fairly similar to the static VPN case. From the point of view of the gateway, a mobile VPN will generate many more tunnels, each of which moving a smaller quantity of data. Also, in a mobile VPN there is much higher connect/disconnect activity.

In both cases the VPN gateways may require enormous encryption throughput. Using AES-MS on the gateways may enable these requirements. Implementation details may vary a little for both cases of VPN. In *static* VPNs the keys are mostly static. This means that a single trusted key could be used to encrypt multiple communications inside a single *gateway→gateway* channel. From the point of view of a multiple-stream encryption core this has the benefit that the key need not be replicated. But this would imply that more ports are needed into the key register. Adding simple circuitry, it is possible to read the register only once and route the key segments to the corresponding encryption engine

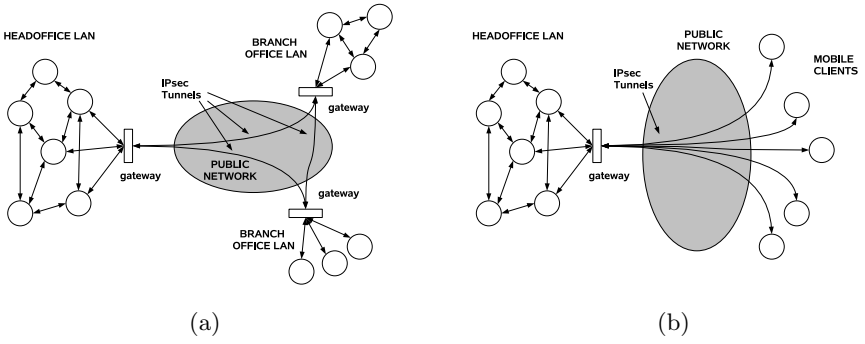


Fig. 2. Sample Architecture for a static VPN (a) and for a mobile VPN (b). In both cases the gateways may need to support high encryption throughput.

without increasing the number of ports. For an architecture in which the encryptions proceed synchronously this technique is trivial; however, in our case, multiple encryptions are performed in parallel but in different iterations. A different strategy is needed in order to maintain the *read only once* property. There are two ways to proceed. One option is to store each segment of the key in a different register. Every engine reads the corresponding key segment from the corresponding register every cycle. Alternatively, we can reduce the number of reads by having the segments read once and then routed to the engine through an appropriate number of latches.

These techniques are easily implemented in ASIC technology; however, when using FPGAs there are additional constraints. For our AES-MS implementation on the Virtex2P this optimization was not readily available due to fact that the base implementation [7] is already optimized to store the full key register in a single BlockRAM using both available ports. However, it may be possible to implement this technique using multiple BlockRAMs of 128 bits. This would then allow to store the complete key schedule and to access the portions independently. However, this particular implementation also consumes many more BlockRAMs, a feature which is undesirable.

In the case of a mobile VPN the technique of sharing the key register is unlikely to result in any benefits. In this environment every client is associated to a different tunnel and each tunnel has its own key, so the gateway cannot share them. Nevertheless, making use of multiple streams is still effective as the aggregate bandwidth of all streams may be very large and serializing the encryption of the packets could otherwise result in network communication degradation.

6.2 Secure File Transfers

When citing VPN we commented that having multiple streams is a key condition to enable our vectorized AES implementation. We now present a particular but still common scenario in which having an AES-MS core can greatly benefit the user.

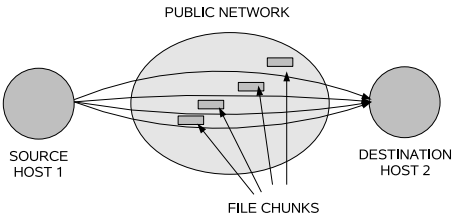


Fig. 3. Parallel file transfer using multiple channels by subdivision of a file into multiple chunks

Transferring files among computers is one of the most common tasks happening on the internet. In general, bulk file transfers can take a long time as they may consist of very large files such as backups, media files, software distributions, etc., being sent to some remote computer. To avoid serialization in this scenario we propose to implement a specialized transfer protocol that opens several tunnels and encrypts multiple parts simultaneously. A single file is subdivided into chunks and sent as multiple files through different channels. This could be done, for example, on modified versions of the `scp` or `sftp` protocols. Figure 3 shows how this would look in the case of a parallel transfer of a single file subdivided into chunks. AES-MS can then be used to accelerate the whole operation.

6.3 Vector Programming Models

So far we have mentioned the application of our technique to gateways in VPN environments but have not commented about the architecture of the gateway itself. There are various levels in which the multiple-stream technique can be implemented. In a pure network device implementation it could be a hardware-only implementation. In this case the gateway just requires a peripheral board with the encryption engine, but this comes at the cost of versatility. The gateway can also be implemented in a higher level using a special programming interface to the device.

Vector architectures provide a special ISA interface in which vector registers can be manipulated as regular registers. The addition of vector loads and stores allows the memory controller to efficiently schedule memory access instructions and better exploit memory bandwidth. Our multiple-stream interface follows an equivalent goal. From the programmers point of view, the AES-MS engine may be programmed as a vector device. Sending multiple unrelated files through input/output channels in a single system call is known as *Vector I/O* or *scatter/gather*. In Figure 4 we show how multiple streams could be encrypted using scatter/gather. The key element of *scatter/gather* is a data structure that holds a vector of data buffers and a corresponding vector with the sizes of each data buffer. The system then reads this data structure and schedules the I/O accesses to the different data buffers in order to maximize system performance. In the

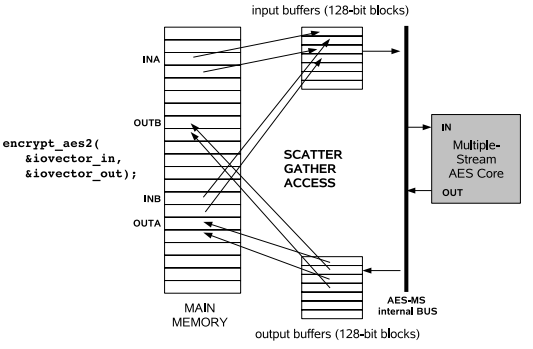


Fig. 4. A simplified Scatter-Gather Interface to Multiple-Stream AES

example, an AES-MS core with two streams is about to process two input data buffers: inA and inB. The system reads the two buffers and interleaves them in blocks of 128 bits. Once this interleaving is done the joint stream can be fed to the AES-MS core for processing. After encryption, the procedure is inverted to store the encrypted data in the corresponding output buffers.

7 Conclusions

In this paper we have described an AES unit capable of processing multiple data streams. Like Vector Engines, our AES unit uses vectors of data to efficiently exploit the external IO bandwidth. The proposed technique can be used to improve throughput in important scenarios such as Virtual Private Networks (VPN) or secure file transfer where large quantities of data are being transferred. We have presented characteristics of the design and proposed a possible programming interface together with possible system architectures for using the core as a coprocessor. The use of the Molen paradigm and the systems reconfigurability, allows to extrapolate these results to other encryption cores. Also, the flexible and modular structure of the used multi-stream AES core allows for an easy integration of additional processing streams, if a higher bandwidth IO bus is used. The bandwidth is being limited by the IO bus and its conservative frequency value, used in the implemented prototype.

Acknowledgments

This work was supported by the HiPEAC European Network of Excellence under contract IST-004408, by the Ministerio de Educación y Ciencia of Spain under contract TIN–2004–07739–C02–01 and by the Portuguese FCT–Fundação para a Ciência e Tecnologia.

References

1. Page, D., Smart, N.P.: Parallel cryptography arithmetic using a redundant montgomery representation 53, 1474–1482 (2004)
2. Crandall, R., Klivington, J.: Vector implementation of multiprecision arithmetic. Technical report, Apple Computer Inc (1999)
3. Bhaskar, R., Dubey, P.K., Kumar, V., Rudra, A.: Efficient Glois Field Arithmetic on SIMD Architectures. In: Proc. of the 15th Annual ACM Symp. on Parallel Algorithms and Architectures, pp. 256–257 (June 2003)
4. Dixon, B., Lenstra, A.K.: Massively Parallel Elliptic Curve Factoring. In: Proc. of the Workshop on the Theory and Application of of Cryptographic Techniques, pp. 183–193 (1992)
5. Vassiliadis, S., Wong, S., Gaydadjiev, G., Bertels, K., Kuzmanov, G., Panainte, E.M.: The MOLEN Polymorphic Processor 53(11), 1363–1375 (November 2004)
6. Vassiliadis, S., Gaydadjiev, G.N., Bertels, K., Panainte, E.M.: The Molen Programming Paradigm. In: Proceedings of the Third International Workshop on Systems, Architectures, Modeling, and Simulation, pp. 1–10 (July 2003)
7. Chaves, R., Kuzmanov, G., Vassiliadis, S., Sousa, L.: Reconfigurable Memory Based AES Co-Processor. In: Proc. of the 13th Reconfigurable Architectures Workshop, IPDPS (January 2006)
8. Kuzmanov, G., Gaydadjiev, G.N., Vassiliadis, S.: The MOLEN Processor Prototype. In: Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2004), pp. 296–299 (April 2004)
9. HTX Electrical and Operational Profile. Technical report, The HyperTransport Consortium
10. HyperTransport HTX: Extending Hypertransport Interconnect Leadership. Presentation, The HyperTransport Consortium (2007)
11. PCI-X 2.0 Overview. Presentation, PCI SIG
12. PCI-SIG - PCI Express Base 2.0 Specification. Technical report, PCI SIG

A Matrix Inversion Method with YML/OmniRPC on a Large Scale Platform

Maxime Hugues and Serge G. Petiton

LIFL, University of Science and Technology of Lille,
59655 Villeneuve d'Ascq, France
{maxime.hugues,serge.petiton}@lifl.fr

Abstract. YML is a dedicated framework to develop and run parallel applications over a large scale middleware. This framework makes easier the use of a grid and provides a high level programming tool. It is independent from middlewares and users are not in charge to manage communications. In consequence, it introduces a new level of communications and it generates an overhead. In this paper, we proposed to show the overhead of YML is tolerable in comparison to a direct use of a middleware. This is based on a matrix inversion method and a large scale platform, Grid'5000.

1 Introduction

Intensive numerical applications like simulation, DNA decoding, climate prediction are parallelised and distributed. Most of those experiments are made by expert scientists of various domains on grids. The main difficulty for them is that each grid has its own properties and different middlewares are deployed on them. Therefore, scientist users must adapted the code to each middleware and this induces a waste of time. The Grid complexity requires a high level programming tool to hide all process to users.

YML [1] is one workflow solution. It is developed at the University of Versailles by the Nahid Emad's team. This framework is dedicated to develop and run parallel applications over large scale middleware. A workflow language named *YvetteML* is used by YML to describe parallelism of each application. YML provides a compiler and a just-in-time scheduler which allows to manage the execution of parallel applications. This transparent management allows to hide numerous communications and code coupling for complex applications. However, to make YML independent from middlewares, an additional communication level is necessary to link the just-in-time scheduler and the selected middleware. The fallout of this layer is that it creates an overhead contrary to a direct use of a client middleware program.

We proposed to evaluate the overhead of YML in different cases. The second section presents motivations of this paper. The third section introduces the Grid'5000 experimental platform, gives an overview of YML Framework and the grid middleware OmniRPC. Then, the Block-based Gauss-Jordan application

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 95–108, 2008.

will be quickly defined and explained. YML experiments and results are presented and analysed in the fifth section. Finally, we conclude and present our future work in the sixth section.

2 Motivations

Companies and laboratories of various domains are more and more interested in grid computing. But in most of case, they have not the technical knowledge to program a grid. YML offers the possibility to develop and run a parallel application without managing communications and is independent from middlewares. The distributed computing is to speed up computations. So, the performance of a workflow framework is an important point and it has not to introduce a significant overhead. In this paper, we propose to estimate and compare the YML overhead with OmniRPC, a cluster/grid middleware. Some experiments are proposed and based on a classical algorithm of matrix inversion, the block-based Gauss-Jordan method. This algorithm offers task dependencies and a lot of communications which are keys of performances of a grid computation. Experiments are done on Grid'5000, a French large scale infrastructure for grid research and experiments. A cluster of 101 heterogeneous nodes is emulated on Grid'5000 to begin. Secondly, we want to observe the management of the computation resources when they are not enough numerous for the number of computation tasks. A cluster emulation of 10 nodes is done with the same applications. However, most of computing resources are distributed across a city, a country or the world like our platform distributed over three different sites (Lille and Orsay in France, Tsukuba in Japan). We proposed to emulate this case on Grid'5000 with heterogeneous networks and heterogeneous computing nodes over five sites geographically. Nevertheless, complex applications have a huge amount of data. These applications use out-of-core techniques. The last experiment is done with an OmniRPC program which uses out-of-core. This program is taken as referent to evaluate the overhead generated by YML.

3 Platform and Environment

In a first step, the GRID'5000 platform is presented, followed by the YML framework, OmniRPC middleware, and to finish by the block-based Gauss-Jordan matrix application.

3.1 GRID'5000 Platform

Grid'5000 [2] is a large scale infrastructure for grid research. It is composed of nine geographically distributed clusters and each one has between 100 to 1000 heterogeneous nodes. This cluster of clusters is interconnected by the French national research network RENATER. Grid'5000 provides reconfiguration and monitoring tools to find out grid issues. This platform allows users to make

reservation, reconfiguration, run preparation and run experiment by using OAR and Kadeploy for nodes reservation and deployment of specific environment which built by user. Grid'5000 is used to investigate issues at different levels of the grid. This includes network protocols, middleware, fault tolerance, parallel/distributed programming, scheduling and issues in performance.

3.2 YML Framework

YML [3] is a framework dedicated to develop and run parallel applications on grids and peer to peer middleware. It is composed of a compiler and a just-in-time scheduler which manages tasks and data dependencies between components [4]. This framework implies a lot of data exchange through the network. To provide and take over data to each component on demand, there is the Data Repository server dedicated. Moreover, YML is independent from the middleware by using an adaptation layer called back-end, see the figure 1.

To describe applications and their executions, YML includes a workflow language called *YvetteML*. The development of an YML application is made using components approach. *YvetteML* components are described using XML and they are three of kinds:

– Abstract component: an abstract component defines the communication interface with the other components. This definition gives the name and the communication channels with other components. Each channel corresponds to a data in input, in output or both and is typed. This component is used in the code generation step and to create the graph.

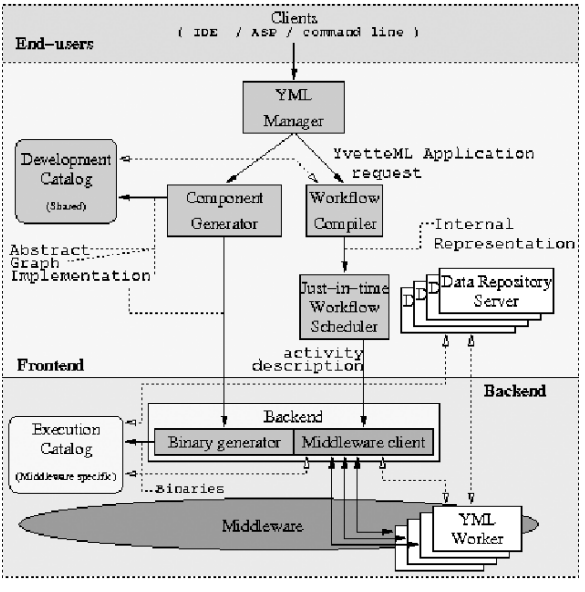


Fig. 1. YML design

- Implementation component: an implementation component is the implementation of an abstract component. It provides the description of computations The implementation is done by using common language like C or C++. They can have several implementations for a same abstract component.
- Graph component: a graph component carries a graph expressed in *YvetteML* instead of a description of computation. It provides the parallel and sequential parts of an application and the synchronize events between dependent components.

Moreover, those three components are independent of middlewares. So, to use an application on another grid with a different middleware, the scientist user has just to compile each component for the middleware of his choice.

3.3 OmniRPC

OmniRPC [5] is a thread-safe remote procedure call (RPC) system, based on Ninf [6], for cluster and grid environment. It supports typical master/worker grid applications. Workers are listed in a XML file named as the host file. For each host, the maximum number of job, the path of OmniRPC, the connection protocol (ssh, rsh) and the user can be defined. An OmniRPC application contains a client program which calls remote procedures through the OmniRPC agent. Remote libraries which contain the remote procedures are executed by the remote computation hosts. Remote libraries are implemented like a executable program which contains a network stub routine as its main routine. The declaration of a remote function of remote library is defined by an interface in the Ninf interface definition language (IDL). The implementation can be written in familiar scientific computation language like FORTRAN, C or C++.

3.4 Block-Based Gauss-Jordan Matrix Inversion

One of the most classical methods for dense matrix inversion is the block-based Gauss-Jordan algorithm [7]. Let A and B be two squares matrices of dimension N, partitioned into $(p \times p)$ blocks of dimension n. Let B be the inverted matrix of A, progressively built. Each of the p steps has three parts (see (1), (2), (3) in the corresponding algorithm 1).

The first part is to invert the pivot block. In the second part, $2(p-1)$ blocks product and finally $(p-1)^2$ blocks triadic are computed. $(p-1)^2$ processors are necessary for computation and each loop **'For'** is executed in parallel.

It is necessary to take into account task dependencies. The figure 2 shows the intra-step and inter-steps parallelism. At each step, the loops (1) and (2) depend on the computation of the inverse block B_{kk} and the loop (3) partially depends on (1) and (2). Then, all matrix products in the loops (1) and (2) are independent tasks and are executed in parallel. The loop (3) is executed in parallel too, because it can start without the complete end of (1) and (2). At the step 2, the computation of the blocks represented by dark squares is not finished. The small numbers in squares represent the computation of the

Algorithm 1. The block-based Gauss-Jordan matrix inversion

Input: A (partitioned into $p \times p$ blocks)
Output: $B = A^{-1}$
  For $k = 0$ to $p - 1$
    $B_{kk} = {A^{-1}}_{kk}$
    For $i = k + 1$ to $p - 1$ **(1)**
      $A_{ki} = B_{kk} \times A_{ki}$
    End For
    For $i = 0$ to $p - 1$ **(2)**
      If $(i \neq k)$
        $B_{ik} = -A_{ik} \times B_{kk}$
      End If
      If $(i < k)$
        $B_{ki} = B_{kk} \times B_{ki}$
      End If
    End For
    For $i = 0$ to $p - 1$ **(3)**
      If $(i \neq k)$
        For $j = k + 1$ to $p - 1$
          $A_{ij} = A_{ij} - A_{ik} \times A_{kj}$
        End For
        For $j = 0$ to $k - 1$
          $B_{ij} = B_{ij} - A_{ik} \times B_{kj}$
        End For
      End If
    End For
  End For

step 3. Our implementation of the Gauss-Jordan algorithm only use the intra-step parallelism. This method is implemented in a client OmniRPC program and in a YML program

4 Experiments

The experiments are done with different architectures of clusters. The table 1 gives the description of the resources that we used. The sites are interconnected by a heterogeneous gigabit ethernet. Before experiments, a node is reserved and a minimal debian is deployed on it to build a dedicated environment. OmniRPC, YML framework and the necessary libraries are installed on it. Then, the environment is recorded on each necessary site for next deployments. The dedicated environment is universal, so it is not necessary to rebuild it on each site to take into account their particularities. The next step is to reserve the required nodes on the grid for the experiment. A shell script is specified to deploy the dedicated environment, prepare the host file which contains the list of computation nodes, launch experiments and get results.

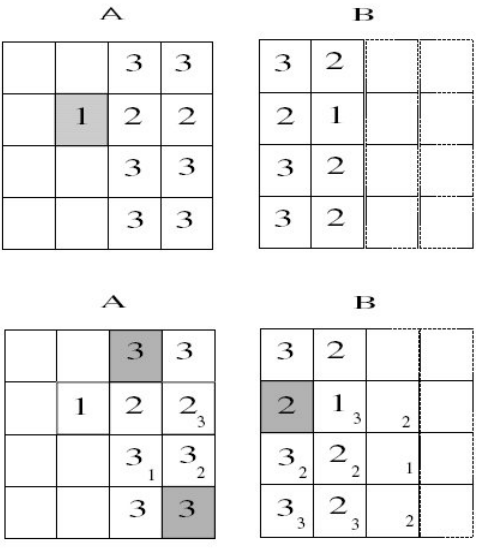


Fig. 2. Intra-step and inter-steps dependencies

Table 1. Computational nodes of Grid'5000

Site	Nodes	CPU/Memory
Nancy	120	2x DC INTEL xeon, 1.6GHz/2GB
Nancy	47	2x AMD64 opteron, 2GHz/2GB
Orsay	216	2 x AMD64 Opteron, 2.0GHz/2GB
Lyon	70	2 x AMD64 Opteron, 2.4GHz/2GB
Sophia	56	2 x DC AMD64 Opteron, 2.2GHz/4GB
Rennes	99	2 x AMD64 Opteron, 2.0GHz/2GB
Rennes	64	2 x AMD64 Opteron, 2.2GHz/2GB

Moreover, one more node is deployed for each experiment. It is not taken into account in the number of deployed nodes, only computation nodes are taken into account. This node plays the role of the client/server and contains the host file required by YML and OmniRPC. The secure shell (ssh) is defined in the host file for communications between the master and workers. In the host file, the maximum number of jobs is set to two because all nodes are dual-processors. The multiplicity of cores are not taken into account in this number. To have a correct estimation of the YML overhead the same parallelism and operations are described in OmniRPC and YML. Four components are defined to implement the block-based Gauss-Jordan algorithm:

1. inversion: to inverse one matrix block
2. prodMat: to compute the two blocks product
3. mProdMat: to compute the negative of two blocks product

4. ProdDiff: to compute the difference between one block and a block matrix product

For OmniRPC experiments, the remote libraries with four components are registered on all computation nodes after the deployment. Then, the executable program takes an argument the number of blocks and it is launched for each value. For YML experiments, abstract and implementation components are compiled and saved in the dedicated environment. A graph component is defined for each number of blocks. After the deployment, all graph components are copied and compiled on the client node. Then, the Data Repository Server is started and the scheduler is launched for each compiled graph component. The first experiment is to emulate a cluster on Grid'5000. 101 nodes are reserved on two clusters of Nancy with mainly dual core Intel Xeon. For a fixed block size (n = 1500) the number of blocks is varied. The condition of $(p - 1)^2$ processors is respected. This increasing variation of the number of blocks goes up with the number of tasks. Thereby, the amount of data dependencies and communications increase too. The execution time of OmniRPC and YML should be made a cubic variation with a constant gap of execution.

5 Results and Analysis

After, the presentation of the dedicated environment and the condition of experiments, a description and an analysis of each experiment is done in this part. The first experiment is an evaluation of the YML overhead in the situation of one cluster composed of heteregenous nodes and networks. The second experiment is similar to the first, but the computation resources are insufficient for the algorithm of Gauss-Jordan. The third experiment is in case of a cluster of clusters distributed geographically. The last section is experiments with an OmniRPC program which uses out-of-core taken as referent to evaluate the overhead of YML.

5.1 Cluster of 101 Nodes

YML overhead is first studied on a single cluster : one geographic site, with heterogeneity of nodes and networks. The first experiment is an emulation on Grid'5000 of a cluster composed of 101 nodes. The Xeon processors are the most numerous nodes. So, the cluster offers 202 processors that satisfy the condition of $(p-1)^2$ processors required by the block-based Gauss-Jordan method. But this condition is available until p equal 15, beyond the computation resources are insufficient. The block size is fixed and for several number of blocks the experiment is done.

The table 2 shows that the overhead is in relation with the number of blocks. More the number of blocks is important, thereof the number of tasks because there are correlated, more the overhead increases. Firstly, this evolution of the overhead comes from in part of the resolution method of dependencies. The YML

Table 2. Time of execution for a cluster of 101 nodes, with block size = 1500

p	Number of tasks	OmniRPC	YML + Backend OmniRPC	Overhead
2	8	281 s	344 s	22.41 %
3	27	487 s	559 s	14.78 %
4	64	712 s	914 s	28.37 %
5	125	965 s	1359 s	40.82 %
6	216	1250 s	2070 s	65.60 %
7	343	1575 s	3103 s	97.01 %
8	512	2102 s	5008 s	138.24 %

Table 3. Time of execution for a cluster of 101 nodes, with block size = 1000

p	Number of tasks	OmniRPC	YML + Backend OmniRPC	Overhead
2	8	108 s	132 s	22.22 %
3	27	165 s	236 s	43.03 %
4	64	242 s	323 s	33.47 %
5	125	328 s	461 s	40.54 %
6	216	425 s	653 s	53.64 %
7	343	541 s	905 s	67.28 %
8	512	676 s	1427 s	111.09 %

scheduler resolves task dependencies at run-time, in opposition to OmniRPC which knows dependencies at compile-time. In an OmniRPC program, the user defines the tasks which have to be waited and launched. In a YML program, the program is translated into a workflow, then the scheduler reads the workflow and launches the tasks without knowing which will end first. The main difficulty for YML is to settle dependencies when it comes at the same time. Moreover, YML has a centralized approach and manages dependencies and data exchange.

Secondly, in this experiment the client/server node has globally the same configuration than the computing nodes. YML has a workload more important than an OmniRPC application. The fallout is the time of execution of an YML program is longer. Although, the overhead stays tolerable until p equal 6, with an overhead of 65 % for 216 tasks.

The execution time of a job on a node play an important role in the overhead of YML and the workload of the scheduler. To reduce the execution on a node and to show this phenomenon, the block size is successively fixed at 1000 and 500. In consequence, a computing node is going to be less loaded and the YML scheduler will be more requested to resolve dependencies at a same time. Furthermore, the data repository server will be more requested to deliver and receive data. So, in this case the overhead should be more important than the first experiment with the block size fixed at 1500.

The first observation is the overhead for a block size of 1000 is less important than a block size of 1500, see table 3. This comes from YML which imports and exports data on the hard disk. If the block size is decreased, the time to read and write the data is shorter. The amount of data to treat between a block

Table 4. Time of execution for a cluster of 101 nodes, with block size = 500

p	Number of tasks	OmniRPC	YML + Backend OmniRPC	Overhead
2	8	41 s	48 s	17.07 %
3	27	50 s	63 s	26.00 %
4	64	58 s	82 s	41.37 %
5	125	75 s	125 s	66.66 %
6	216	83 s	207 s	150.39 %
7	343	103 s	277 s	168.93 %
8	512	130 s	379 s	191.53 %

size of 1000 and 1500 has a ratio of 1/2. The computation time for a block is approximately the same, but the time to write/read data on hard disk is different. So, the access time to the disk gained by YML decreases the overhead. But for a block size of 500, the overhead is more important, see table 4 and figure 3. Because the computation time of a block on a node is shorter, then the scheduler of YML is more requested to resolve dependencies. Furthermore, the data repository server is more requested too for the data exchanges.

5.2 Cluster of 10 Nodes

To decrease the use of the YML scheduler and the data repository server, a second experiment is done with few computation resources. The use case is the same than the first experiment: one cluster, one geographic site, with heterogeneity of nodes and networks. The second experiment is an emulation of a cluster composed of 10 nodes. So, the cluster offers 20 processors that satisfy the condition of $(p - 1)^2$ processors required by the block-based Gauss-Jordan method. But this condition is available until p equal 5, beyond the computation resources are insufficient. The block size is fixed and for several number of blocks the experiment is done.

First observation, the execution times for a cluster of 10 nodes are less important than a cluster of 101 nodes until p equal 5 for a block size of 1500, see table 5. When OmniRPC starts, it builds a database which contains the computing nodes. In this experiment, the nodes are less than the experiment with 101 nodes. So, the database is built faster. For p from 6 to 8, the execution times have an additional delay. Because the computing resources are not sufficient beyond p equal 5. To execute the next task, it is required to expect a free node.

Second observation, the overhead is not very important. It is between 10 and 20 percent for p from 2 to 5. The overhead is lower than the results of the experiment of 101 nodes on one cluster. The number of tasks is more important than the computer resources available. The fallout is the scheduler of YML is less requested to solve the data dependencies at a same moment. The dependencies are solved faster. The data repository server is less used to deliver and receive the data. This explains that the execution times are approximately the same as the experiment of 101 nodes for $p > 5$.

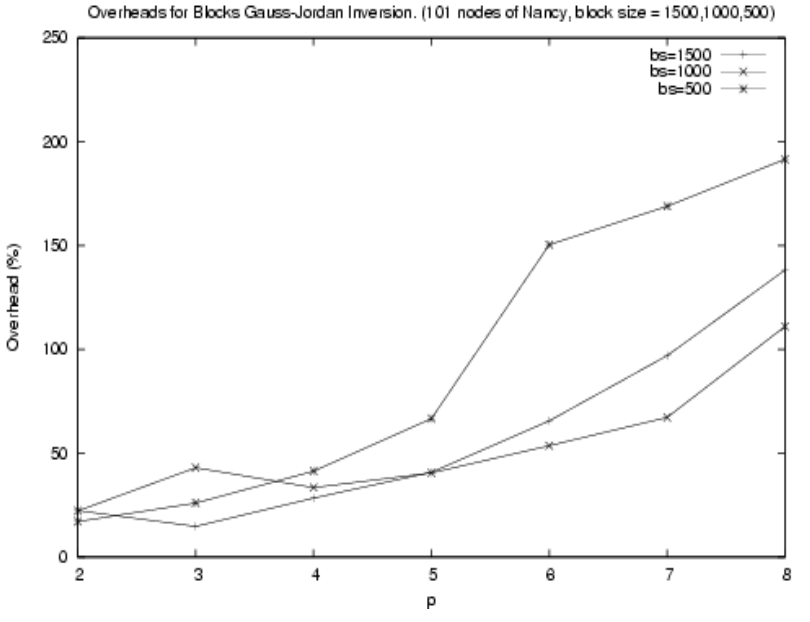


Fig. 3. Overhead for different blocks size on a 101 nodes cluster on Nancy

5.3 Cluster of Clusters

After the evaluation of the YML overhead in the case of one cluster. We want to evaluate the overhead in the case of a cluster of clusters like our platform distributed between Lille, Orsay in France and Tsukuba in Japan. This experiment is an emulation on Grid'5000 of a cluster of 101 nodes distributed over 5 geographic site (Nancy, Orsay, Sophia, Lyon, Rennes) and over 6 clusters. The nodes are distributed in an homogeneous way, 17 nodes per site, excepted Rennes which has 34 nodes distributed over two clusters. The site of Nancy counts 16 nodes and the node which plays the role of client/server. So, the cluster offers 202 processors that satisfy the condition of $(p - 1)^2$ processors required by the block-based Gauss-Jordan method. But this condition is available until p equal 15, beyond the computation resources are insufficient.

Table 5. Time of execution for a cluster of 10 nodes, with block size = 1500

p	Number of tasks	OmniRPC	YML + Backend OmniRPC	Overhead
2	8	203 s	247 s	21.67 %
3	27	406 s	453 s	11.57 %
4	64	660 s	740 s	12.12 %
5	125	982 s	1204 s	22.60 %
6	216	1454 s	1945 s	50.00 %
7	343	1874 s	3073 s	63.98 %
8	512	2600 s	6076 s	133.69 %

Table 6. Time of execution for a cluster of clusters of 101 nodes, with block size = 1500

p	Number of tasks	OmniRPC	YML + Backend OmniRPC	Overhead
2	8	307 s	361 s	17.48 %
3	27	506 s	578 s	14.22 %
4	64	727 s	910 s	25.17 %
5	125	1193 s	1487 s	24.64 %
6	216	1659 s	2702 s	62.86 %
7	343	2258 s	3164 s	40.12 %
8	512	2921 s	5836 s	99.79 %

The heterogeneity of networks adds time for communications between the computing nodes and the server. The difference of execution times for OmniRPC between the table 6 and the table 2 are between 20 and 800s. In the case of YML, the difference of execution times are less significative, between 20 and 630s. The overhead of YML in this configuration of cluster of clusters are acceptable, between 14 and 63 percent. Because the communication times are important, so the scheduler and the data repository server of YML are less requested.

5.4 Out-of-Core Gauss-Jordan

The previous evaluations of the overhead are made in the worst case. After each component call, YML reads and writes data on the hard disk. However, our OmniRPC program stores data in live memory. So, the access time to the data for YML is more important than OmniRPC. In this section, the evaluation of the overhead is made with the same YML program and an OmniRPC program which uses out-of-core. It is very important to notice that the OmniRPC program with out-of-core is not a version of Gauss-Jordan with out-of-core. The program only reads and writes data at each step of the Gauss-Jordan method, like [8]. This evaluation is firstly done with the same condition than the first experiment: one cluster, with heteregenous nodes and networks. This cluster is based on Nancy

Table 7. Time of execution of Gauss-Jordan out-of-core on a cluster of 101 nodes, with block size = 1500

p	Number of tasks	OmniRPC out-of-core	YML + Backend OmniRPC	Overhead
2	8	321 s	344 s	7.16 %
3	27	539 s	559 s	3.71 %
4	64	851 s	914 s	7.40 %
5	125	1193 s	1359 s	13.91 %
6	216	1625 s	2070 s	27.38 %
7	343	2049 s	3103 s	51.44 %
8	512	2564 s	5008 s	95.31 %

Table 8. Time of execution of Gauss-Jordan out-of-core on a cluster of clusters, with block size = 1500

p	Number of tasks	OmniRPC Out-of-Core	YML + Backend OmniRPC	Overhead
2	8	329 s	361 s	9.72 %
3	27	590 s	578 s	-2.03 %
4	64	947 s	910 s	-3.90 %
5	125	1412 s	1487 s	5.31 %
6	216	2080 s	2702 s	29.90 %
7	343	2796 s	3164 s	13.16 %
8	512	3605 s	5836 s	61.88 %

and has 101 nodes. So, the cluster offers 202 processors that satisfy the condition of $(p - 1)^2$ processors required by the block-based Gauss-Jordan method. But this condition is available until p equal 15, beyond the computation resources are insufficient. To compare these new results with the first experiment, the block size is fixed at 1500 and for several number of blocks the experiment is done.

In the table 7, the execution times of OmniRPC are higher than the results in the table 2. This increase comes from the location of data. In the case of our Gauss-Jordan out-of-core, data are located on the hard disk. They are loaded in the main memory when they are going to be used in the step k. The access time to data located on the hard disk is more important than the access time to data located in main memory. The figure 4 shows the overhead is smaller

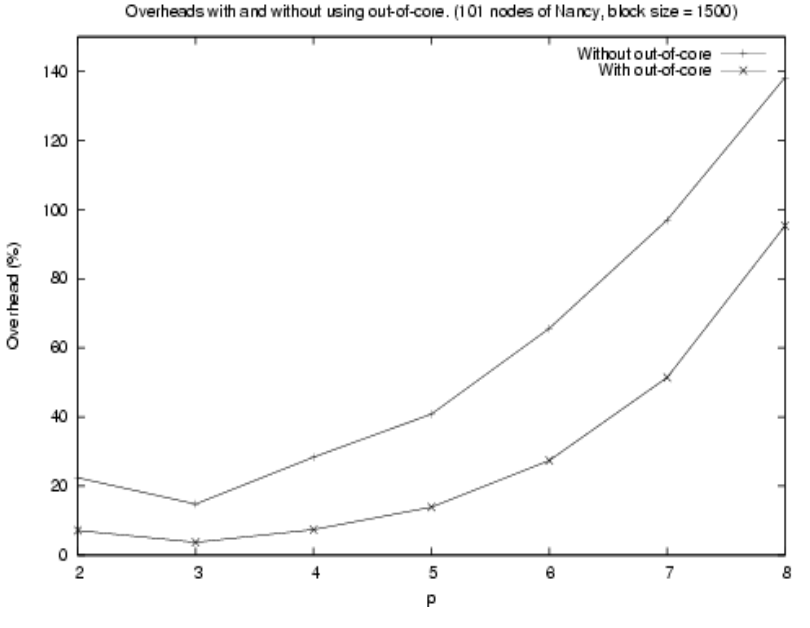


Fig. 4. Overheads with and without using out-of-core

with an OmniRPC program which uses out-of-core. The difference between the overheads varies from 11.07% for p=3 to 45.47% for p=7.

The overhead of YML is less in the case where the Gauss-Jordan algorithm stores data on the hard disk. Moreover, we could see the cluster of clusters configuration decreases the overhead. The next experiment is also to combine this configuration with our OmniRPC out-of-core program. The goal of this experiment is to see if the overhead of YML can be decreased in the case of a large scale distributed application.

There is a gain for p equal 3 and 4. YML is faster than our OmniRPC program for this two cases. The scheduler of YML solves dependencies at the runtime. In opposition, the dependencies of our OmniRPC programme are specified by the coder when he writes the application. So, the OmniRPC programme have not to solve dependencies but only to synchronize the dependent tasks with some wait and call functions.

6 Conclusion

In this work, we have evaluated the overhead of YML compared to an OmniRPC programme with different architecture of cluster and in two cases of data storage. In the first case, the OmniRPC program of reference stored data in the main memory. So, the OmniRPC program accessed faster to data than YML which had to load and unload data from hard disk at each component call. The experiments have showed that the overhead of YML was not important when the scheduler is not overloaded to solve the data dependencies, between 10 to 60% for a maximum of 512 tasks. In the second case, we have taken as referent an OmniRPC program which stores data on the hard disk. Because a lot of complex programs which have an important amount of data use out-of-core techniques. In this context, the overhead of YML is under 95% for the same experiments of the first case. Furthermore, with an architecture of cluster of clusters, YML can be faster than our OmniRPC program. Even though, 100 or 150% of overhead are high values in some cases. It is important to notice that YML allows to reuse the code for different middlewares. An YML application has not to be adapt for an other platform which uses a different middleware. Moreover, it is not necessary to manage communications like an MPI program. And the end user can improve parts of code to decrease the overhead.

To complete this work, we plan to evaluate the overhead with huge matrix and clusters which have more processors. YML does not integrate a data persistence system and multicore management at the moment. But we could evaluate the overhead with an emulation of the data persistence. The matrix blocks would be regenerated on the nodes. And the scheduler of YML will be improve to have less overhead. In the future, YML could use many middlewares at the same moment with a multi-backend support. This feature is in prevision to program hybrid clusters with a high level programming tool. For the moment, the framework YML is a performing tool for average application and has a tolerable overhead. It allows to program easily a grid without manage communications,

it supports two middlewares and soon three with the arrival of the backend for condor.

Acknowledgment

In addition, we thank a lot Olivier Delannoy and Nahid Emad from PRiSM, Laurent Choy and Mitsuhisa Sato from the University of Tsukuba for their help and collaboration on YML and OmniRPC. We thank a lot INRIA for their help on Grid'5000.

References

1. YML Project Page, `http://yml.prism.uvsq.fr`
2. Cappello, F., Caron, E., Daydé, M.J., Desprez, F., Jégou, Y., Primet, P.V.-B., Jeannot, E., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Quétier, B., Richard, O.: Grid'5000: a large scale and highly reconfigurable grid experimental testbed. In: The 6th IEEE/ACM International Conference on Grid Computing, pp. 99–106 (2005)
3. Delannoy, O., Emad, N., Petiton, S.G.: Workflow Global Computing with YML. In: The 7th IEEE/ACM International Conference on Grid Computing, pp. 25–32 (2006)
4. Petiton, S.: Parallelization on an MIMD computer with real-time Scheduler. In: Aspects of Computation on Asynchronous Parallel Processors. North Holland, Amsterdam (1989)
5. Sato, M., Boku, T., Takahashi, D.: OmniRPC: a Grid RPC ystem for Parallel Programming in Cluster and Grid Environment. In: The 3rd IEEE International Symposium on Cluster Computing and the Grid, p. 206 (2003)
6. Sato, M., Nakada, H., Sekiguchi, S., Matsuoka, S., Nagashima, U., Takagi, H.: Ninf: A Network Based Information Library for Global World-Wide Computing Infrastructure. In: HPCN Europe, pp. 491–502 (1997)
7. Aouad, L.M., Petiton, S., Sato, M.: Grid and Cluster Matrix Computation with Persistent Storage and Out-of-Core Programming. In: The 2005 IEEE International Conference on Cluster Computing, Boston, Massachusetts (September 2005)
8. Melab, N., Talbi, E.-G., Petiton, S.G.: A Parallel Adaptive version of the Block-based Gauss-Jordan Algorithm. In: IPPS/SPDP, pp. 350–354 (1999)

Can We Connect Existing Production Grids into a World Wide Grid?

Peter Kacsuk[1], Attila Kertesz[1], and Tamas Kiss[2]

[1] MTA SZTAKI
1111 Budapest, Kende u. 13
{kacsuk,attila.kertesz}@sztaki.hu
[2] Centre for Parallel Computing, University of Westminster
115 New Cavendish Street, London, W1W 6UW
kisst@wmin.ac.uk

Abstract. The World Wide Web has become a phenomenon that now influences our everyday life in any possible areas and disciplines. This paper investigates how a grid equivalent of the WWW, the World Wide Grid can be created. We define requirements towards a workflow-oriented computational World Wide Grid and propose a solution how current production Grids can be connected in order to form the technical basis of this infrastructure. A meta-broker concept and its utilization to achieve the highest level of parallelism by the created architecture in a user transparent way are explained.

Keywords: Grid interoperability, meta-broker, parallelism, grid workflow, World Wide Grid.

1 Introduction

The goal of this paper is to assess where we are in the road of establishing a scientific, workflow-oriented, computational World Wide Grid (WWG) that is similar to the World Wide Web (WWW) in the sense that anyone can access and use its services according to his needs. If we look at the current trend of how and where grid develops we can see that isolated production grids have been created that are based on different grid technologies that are not interoperable or only in a limited way. One of the biggest challenges of the grid community is to solve the interoperability of these production grids in order to get closer to the idea of WWG.

This paper is intended to show that we are not far from creating a WWG if we make reasonable assumptions how the users would like to use a WWG and reasonable restrictions how such a WWG can be used and for what purposes. The basic assumptions are as follows:

1. We restrict ourselves to a computational grid type WWG
2. The goal of the user is to dynamically collect and use as many grid resources as possible to accelerate his grid application
3. The basic application type the user would run in the WWG is a complex workflow
4. The WWG will be used in the beginning by the scientific community

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 109–122, 2008.

Assumption 1 says that we restrict the first version of the WWG to a computational grid where the size of files used in the WWG is not too large and hence they can efficiently be moved between computational grid resources. It does not exclude the usage of large files in the WWG but it significantly reduces the overall performance of the WWG if many users try to use large files. Obviously, in a longer term the efficient management of large files should also be solved in the WWG but since our goal is to show that a computational WWG is already feasible we neglect this problem in this paper. A follow-up paper will be written to outline the possible solutions for a data-oriented WWG.

Assumption 2 requires that all possible parallelisms of an application should be exploited in the WWG. Users go for the WWG to access and use many resources in parallel to speed up the execution of their complex applications. Section 2 will classify the types of parallelisms that can be achieved in the WWG and later it will be shown how such parallelisms can actually be utilized in the WWG.

Assumption 3 might require some more explanations then the first two assumptions. Why workflow applications are so interesting for the WWG? The workflow concept abstracts a collection of services (tasks) that can be executed in a partially ordered way and hence it is general enough to include as a special case any other types of applications. As a result, workflows could be considered as the most generic type of applications including any other types as special cases.

Assumption 4 is based on the current usage scenario of large e-science grids. In order to establish a WWG that is used by the general public or by businesses, it requires a significant improvement in the development of a grid market model. If only the scientific community will use the first WWG, then the grid market model could be much simpler than a real commercial one. Since our interest is to establish a WWG as soon as possible, it is better to start with a scientific WWG and later extend it to other directions.

At this point many readers could say that these assumptions are too restrictive and it is not worth defining and creating a WWG that can support only these kinds of applications and goals. We would argue that we cannot create at once the ultimate WWG. The WWW is much more complex today as it was in its initial stages and in the beginning it was used only for scientific purposes and only later it was extended towards the commercial world. Once we established an infrastructure that is useful and works, there are plenty of opportunities afterwards to improve and extend that system in the future. Even this restricted version of the WWG that is suggested in this paper can be used to support much more applications than we can dream today. The establishment of such a WWG could tremendously widen the user community of the grid and would significantly accelerate the take-up of grid technology world-wide and would lead later to a WWG usable for the whole population including commercial services as well.

In this paper we identify the main steps of creating a WWG system according to the assumptions mentioned above, and describe in detail the first of these steps that provides the basic interconnection and access mechanism of such a WWG system. Section 2 analyses the reasons of the success of the WWW and compares the WWW concept with a potential WWG concept. Section 3 classifies the achievable types of parallelisms in a WWG system. Section 4 introduces the concept of meta-broker and shows how the various types of parallelisms can be exploited in the WWG by using the meta-broker concept. Section 5 gives a short overview of related research.

2 Comparison of WWW and WWG

Before starting to explain the technical details of creating such a WWG, it is important to compare the WWW concept and the proposed WWG concept. First of all, let's see why the WWW concept is so popular and successful. There are five main aspects that make WWW so attractive:

1. Services
2. User interface
3. Web search engines
4. Security
5. Interest to use

The original WWW concept was based on the web page services. The idea is that anyone can create and publish web pages and anyone from any client machine can access these published web pages. Another important concept here is that web pages can be linked together and this significantly facilitates the creation and usage of web pages.

The second appealing feature of the WWW is that its user interface is extremely simple and easy-to-use. A simple browser is sufficient to browse web pages no matter where these web pages were created. More than that these browsers are provided as part of the basic services of the client machines' operating systems and hence the user does not have to install any complicated WWW access software, it comes together with the client machine. Web portals help the users to access structured information over a set of web pages.

Web search engines [1][2][3] help the users to discover information in the Web. In fact the Web search engines provide the web information system by discovering relevant web page contents.

The HTTP and HTTPS protocols provide the necessary security mechanism. They require only a single port to be opened on the server machine and hence they can be securely managed. The security concept of the WWW is so reliable that even large banks trust this system and provide financial services through their web portals.

The final aspect of the WWW is the motivation of people to use it or to provide services by it. The WWW is an excellent way of creating communities, accessing information and special (e.g. financial) services and hence people are interested in using the WWW services. At the beginning when commercial exploitation of the WWW was not so apparent, people were interested in creating web pages because in this way they could increase their or their company's visibility. Later when it became clear that there are several business models by which companies can make profit by providing WWW services, the usage of the WWW became even more popular.

After the overview of the main aspects of the WWW let's see how these aspects can also be used to promote the WWG as a success story. In case of the WWG the services are grid resources (computing services, data services, etc.) or higher level grid services (for example, a workflow execution service). It is important that anyone should be able to provide grid services and anyone should be able to access these grid services from any client machine through the WWG. The same way as web pages can be linked together grid services should be linked together, too.

The user interface should be extremely simple and user-friendly, exactly the same way as in case of the WWW. Simple tools like a web browser should be available on every client machine in order to access the WWG services without installing any grid software on the users' machines. However, in the WWG the main objective is to run applications and hence instead of a simple browser, rather a simple application builder is the right GUI. Grid portals should help users to access the WWG services in a co-ordinated way releasing the user from the actual organization of resource collection and orchestration via the WWG.

Grid search engines similar to the Web search engines should help both users and grid services to discover relevant services of the WWG.

The current grid security mechanism is built on the concept of grid certificates and VOs. Although scientific papers always emphasize the importance of creating dynamic VOs, in practice VOs are quite static and their creation is a long procedure. This static nature of VOs is one of the reasons why the grid has been developed towards the isolated grids direction and not towards the WWG direction. The current certificate and VO scheme make the usage of the grid much more complicated than the usage of the Web. As a consequence in this respect some revision is necessary if we want to make the WWG really easy-to-use and popular.

Table 1. Comparison of WWW and WWG

	WWW	WWG
Services • Anyone can create and publish • anyone can access	Web pages, web services	Grid resources, computing services, data services, etc.
User interface	Web browsers Web portals	Grid application builder Grid portals
Information discovery mechanism	Web search engines	Grid search engines
Security	HTTP and HTTPS protocols	Revised dynamic VO concept
Interest to join	User: access information and services Provider: Increase own visibility, make money	User: use grid resources (by grid credit) Provider: collect grid credits (later make money)

Finally, the motivation of the usage of WWG should be made tempting for large user and grid service provider communities. Once the usage of WWG is simple enough it will be really attractive for a large user community to access grid resources and services in an "unlimited" way. This will raise a new problem. If there is no limit

of accessing resources and services the whole WWG will collapse due to the huge demand of resources and services. A WWG market model is therefore unavoidable and obligatory from the very beginning in order to attract resource and service providers and to restrict the eagerness of WWG consumers in acquiring resources and services for their applications. A kind of WWG credit system must be introduced where resource and service providers can earn WWG credits and then their community can use these credits to acquire WWG resources and services.

Table 1 summarizes and compares the five main features of the existing WWW and a potential WWG.

If there are so many similarities in the concept of WWW and WWG, then why are we still missing the WWG as a working infrastructure and service? Unfortunately, there are some problems concerning all the five required features of the WWG.

Services and Resources

There are many different production grids based on different grid technologies and middleware and they can not interoperate. As a result the services are not accessible by anyone from anywhere as it would be needed by the WWG concept proposed above. If a user is registered for the VOx of GridA then he cannot use the resources and services of VOy of GridB. Section 4 of this paper will show that based on assumptions 1-3 we can easily solve this problem and create a WWG that satisfies the required criteria.

User Interface

The grid user interface is currently too complicated. In most cases production grids neglect the problem of user interface. They provide only a special command line interface and programming API. It means that there is no user interface standard like the web browser in case of the Web. Different grid middleware requires the usage of different command line interfaces and programming APIs. It means that a user who wants to use several grids (an obvious assumption of the WWG concept) has to learn several grid user interfaces. If the user wants to port a grid application from GridA to GridB he has to re-engineer the application according to the programming API of GridB.

Information Discovery Mechanism

The grid information system and discovery mechanism are not mature enough and the concept of a grid search engine is also missing from current grids. The usage of the information system is very limited in current production grids.

Security

The grid security mechanism is suitable for the rather static VO concept of current production grids but not for the WWG where VOs should be formulated dynamically on demand. As a consequence the concept of VOs should be revised in the framework of the WWG. However, the subject is so big and significant that an independent paper should deal with this issue. One solution could be that only pre-registered and verified applications can be used in the WWG that are taken from certified application registries. Another solution could be the usage of virtualization where the application is distributed inside a virtual machine that can not cause any harm to the executor resource.

Interest to Join
Since the current usage of grid lacks the market concept (everyone can get what he needs without payment) a WWG would lead to the tremendous overload of resources. At least, the introduction of a simplified grid market concept would be necessary to establish a scientific WWG.

The following sections of this paper describe in detail the first of these features and show possible solutions that can be used in order to establish a scientific computational WWG. The analysis and solution of the other listed aspects will follow in future publications. Obviously, when the goal is to create a WWG as soon as possible any proposed solution should be built according to the current situation. It should be accepted as a fact that production grids already exist and they are not willing to give up their freedom of developing their own roadmap and direction. Therefore a WWG concept should be built on the autonomicity and collaboration of these existing production grids.

3 Parallelism in the WWG

As we have seen in the Introduction, assumption 2 says that the goal of the user is to

a. dynamically collect and use as many grid resources as possible
b. in order to accelerate his grid application.

Condition "a" means that the user needs a WWG where resources can be accessed from any production grid that are connected to the WWG no matter what type of grid middleware they are built on. As a result current production grids should be used in an interoperable way even if they are based on different grid middlewares. When this problem is solved any user from any grid can access all the grid resources and services that are connected to the WWG.

Condition "b" requires that both the application and the WWG should be able to support the largest possible classes of parallel execution.

In order to fulfil our requirements, first we examine the impact of parallelism on the grid middleware, and then investigate the problem of grid interoperability. We shall see that both conditions can be fulfilled by the introduction of interoperable grid brokers or by the introduction of a meta-broker.

There are two classes of parallelisms achievable in the WWG:

1. Grid architecture parallelism
2. Grid application parallelism

The grid architecture parallelism can be further divided into two classes according to the usage of various grid resources:

– **Inter-Resource (IrR) parallelism** (parallel usage of several resources) within which we can distinguish:
  - **Inter-Grid (IrG) parallelism** (parallel usage of several resources within several Grids)
  - **Intra-Grid (IaG) parallelism** (parallel usage of several resources in the same grid)

- **Intra-Resource (IaR) parallelism** (usage of parallelism in a single resource having parallel architecture, e.g. cluster)

The current grids typically enable the exploitation of the Intra-Grid and Intra-Resource parallelism but they do not support Inter-Grid parallelism. Even worse, within the same grid, users are restricted to use the resources of a certain VO only where they are accepted as members. The current concept of VOs is strongly against the nature of WWG.

We can distinguish four types of grid application parallelism according to the granularity of tasks to be solved in the grid:

– Single job/service level (SJ)
– Parameter Sweep at job/service level (PSJ)
– Workflow level (WF)
– Parameter Sweep at workflow level (PSWF)

Intra-Resource parallelism can be applied to any types of grid application parallelism if the resource is a multi-processor one (either shared or distributed memory system or even multi-core processor) and the local job manager is able to distribute the parallel components of the application among the processors of the resource.

Single job level parallelism (SJ) can be exploited if the application consists of one job and this job is a parallel (e.g. MPI) one. In this case we can explore **process parallelism** that comes from the parallel execution of processes of the parallel application. Although there are some research projects aiming at the exploitation of Inter-Resource parallelism, SJ parallelism still best fits to the Intra-Resource parallelism where processes of a parallel application can be distributed among the nodes, processors or cores of a parallel resource. If there are N processes inside a parallel job, then the achievable parallelism is O(N).

PS job level parallelism (PSJ) can be exploited if the application consists of one job and this job should be executed with many different parameter sets (this is called **job instance parallelism**). PSJ parallelism fits both to Intra-Resource and Inter-Resource parallelism no matter whether the job is a sequential or parallel one. In the case of a sequential job Intra-Resource parallelism can be exploited by allocating many instances of the same job to a multiprocessor resource and the different job instances are simultaneously executed by different processors of the resource. If a sequential job is to be executed with M parameters, then the achievable parallelism is O(M). If the job is a parallel application with N processes and this should be executed with M parameters, then both process parallelism and job instance parallelism can be exploited and the achievable parallelism is O(MxN). Since both N and M can be in the range of hundreds or thousands, PSJ parallelism can require several thousands of resources and hence it really needs the large number of resources available in the WWG.

Workflow level parallelism (WF) can be exploited if there are parallel branches in the workflow. This is called **workflow branch parallelism** and it fits both to Intra-Resource and Inter-Resource parallelism. However, the amount of parallelism is typically not as big as in case of the PSJ parallelism so in many cases Intra-Grid parallelism is enough to handle it. If some of the jobs in the workflow are parallel (e.g. MPI) jobs, then two levels of parallelism can be exploited simultaneously: process

parallelism and workflow branch parallelism. Different MPI jobs of the different branches of the workflow can be simultaneously executed on different resources (Inter-Resource parallelism) and on each such resource Intra-Resource parallelism can be applied for the parallel execution of the processes of the MPI jobs. If the maximum number of parallel branches in the workflow is B and in every branch there is a parallel application with N processes, then the achievable parallelism is O(BxN).

PS workflow level parallelism (PSWF) can be exploited if the application consists of a workflow and this workflow should be executed with many different parameter sets (this is called **workflow instance parallelism**). In such case three levels of application parallelism can be exploited:

- Workflow instance parallelism (among the several instances of the workflow)
- Workflow branch parallelism (among the branches of every workflow instance)
- Process parallelism (among the processes of a parallel node of a workflow instance)

Notice that job instance parallelism is a special case of workflow instance parallelism when the workflow consists of a single job. From another point of view workflow instance parallelism is a sum of achievable job instance parallelism when the component jobs of a workflow are executed with many parameters sets.

In order to exploit these three levels of parallelism, PSWF parallelism can require thousands or even millions of resources and hence it really needs the large number of resources available in the WWG. PSWF parallelism fits to the Inter-Resource parallelism (both IaG and IrG) and as a result can advantageously be used in the WWG. If the maximum number of parallel branches in the workflow is B and in every branch there is a parallel application with N processes, and the workflow should be executed with M different parameter sets, then the achievable parallelism is O(MxBxN). This is clearly the most demanding type of parallel application concerning the number of required grid resources.

4 Resource Selection to Achieve the Highest Possible Parallelism

After seeing that many resources should be used in a PSJ or PSWF application the next question is how to select the required resources in order to achieve the highest possible parallelism. In a WWG system four models can be distinguished:

– User selected Grid User selected Resource (UGUR)
– User selected Grid Broker selected Resource (UGBR)
– Broker selected Grid User selected Resource (BGUR)
– Broker selected Grid Broker selected Resource (BGBR)

More and more production grids use brokers nowadays inside their own boundaries in order to select the most appropriate resources for execution. However, even if the user is allowed to submit jobs to several grids, it is his responsibility to select between these grids. Brokering at Grid level is not supported in today's production environments. As a result, the BGBR model, that provides the largest freedom of accessing grid resources in the WWG, is hard to realize. The BGBR model means that the users

can access several grids simultaneously and these grids have got brokers. However, the user is not connected directly to a grid broker rather to a new level of brokers called as meta-broker. It is the task of the meta-broker to select the right grid according to the users' requirements as described in the Broker Property Description Language (BPDL) [4]. BPDL is similar to the JSDL [6], but it provides metadata about brokers (grids) and not about resources. Once the grid is selected the meta-broker passes the job and the job description language (RSL [7], JDL [8] or JSDL depending on the actual grid) to the selected grid broker and it will be the task of this broker to select the best resource according to the requirements specified in the job description language (or sometimes called resource specification language). The architecture of the BGBR model can be seen in Fig. 1.

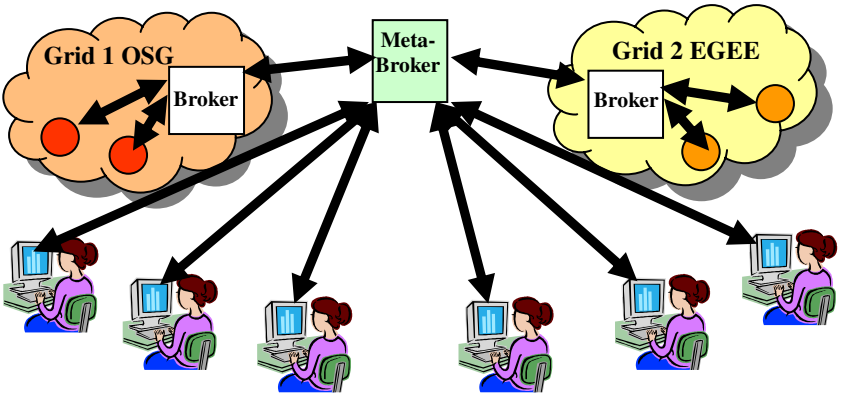


Fig. 1. Grids connected by a meta-broker

The proposed Meta-Broker architecture is described in detail in [5]. In order to make the concept of meta-broker clear a short overview of its architecture concept is given here. Fig. 2 introduces the proposed architecture of the Grid Meta-Broker that enables the users to access resources of different grids through their own brokers. The Translator components are responsible for translating the resource specification language of the user (JSDL) to the language of the selected resource broker. Once a broker will be capable of supporting the JSDL standard [6], the corresponding Translator can be removed from the Meta-Broker. The Invokers are broker-specific components. They communicate with the interconnected brokers, invoking them with job requests and collecting the results. Data handling is also an important task of this component. After the user uploaded the job, proxy and input files to the Meta-Broker, the Matchmaker component tries to find a proper broker for the request. If no good broker was found, the request is rejected, otherwise the JSDL is translated to the language of the selected broker. The responsible Invoker takes care of transferring the necessary files to the selected grid environment. After job submission it stages the output files back, and upgrades the historical data stored in the Information Collector with the logging data of the utilized broker. The core component of the Meta-Broker is responsible for managing the communication (information and data exchange) among the other components. The communication to the outer world is also done by this part through its web-service interface.

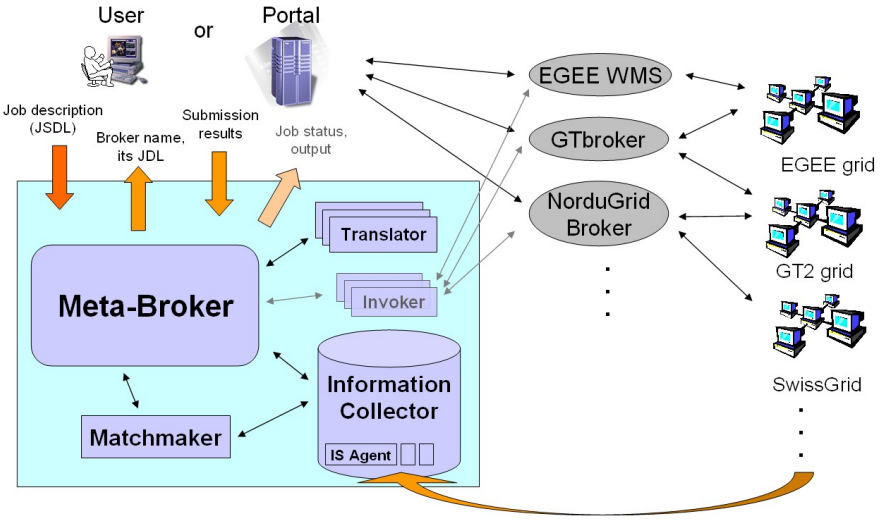


Fig. 2. The Grid Meta-Broker Architecture

In the following we examine how SJ, PSJ, WF and PSWF parallelism can be exploited by the BGBR model and by a meta-broker.

SJ Level Parallelism:
In the BGBR case both the grids and the resources are selected by brokers and hence load-balancing can be achieved between grids and inside grids.

PSJ Level Parallelism:
Both grid and resource selection is done by a broker and hence both Inter-Grid (IrG) and Intra-Grid (IaG) parallelism are automatically provided by brokers. In principle all resources of all connected grids can be used in parallel and in a balanced way.

WF Level Parallelism:
Grid selection is the task of the broker and hence it is the broker's responsibility to explore the Inter-Grid (IrG) parallelism through the workflow branch-parallelism. Resource selection is also the task of the broker and hence it is the broker's responsibility to explore Intra-Grid (IaG) parallelism through workflow branch-parallelism and to explore Intra-Resource (IaR) parallelism by assigning parallel jobs of the workflow to parallel grid resources. Since both the grid and the resources are selected by brokers, load balancing can be achieved between grids and inside grids. However, workflow scheduler support is needed for the broker to exploit workflow branch-parallelism at IrG and IaG level.

PSWF Level Parallelism:
Both grid and resource selections are the tasks of the broker and hence both Inter-Grid (IrG) and Intra-Grid (IaG) parallelism can be exploited both through the workflow

branch-parallelism and workflow instance-parallelism by the broker. Intra-Resource (IaR) parallelism can also be achieved as described in Section 3.

Advantages of the BGBR model are:

- Workflow instance-parallelism can be exploited both at IrG and IaG level.
- Workflow level scheduling support is not needed for the broker either at IrG or IaG level to achieve workflow-instance parallelism.
- Balanced usage of grids and resources are provided by the broker.

As a summary one can say that the most advantageous model to exploit every possible parallelism in a well balanced way is the BGBR model. So if we want to establish an efficient and powerful WWG it should be based on the concept of the BGBR model. Unfortunately, current grid middleware do not support the BGBR model. There is an on-going effort in the GIN VO of OGF to solve grid interoperability, but they concentrate at the moment on the SJ level. Within the framework of the CoreGrid project SZTAKI and UPC work on to design a meta-broker that can efficiently support the BGBR model [9].

Naturally, a single meta-broker would be a bottleneck in the WWG used by many thousands of scientists. Therefore, many uniform meta-brokers should be applied and these should be interconnected in order to realize load-balancing among them. In such a WWG every client can be connected in a standard way to one of the uniform meta-brokers. As a result if we want to build the WWG, the only thing we have to do is to define and implement:

1. The functionality of the meta-brokers
2. The intercommunication protocol of meta-brokers
3. The communication protocol of clients and meta-brokers
4. The standard intercommunication protocol of meta-brokers and grid brokers

The solution for requirements 1 and 3 are already designed and described in [5]. Work is needed for requirements 2 and 4. Notice that the BGBR model can be used even if requirement 4 is not fulfilled.

5 Related Work

The most notable work related to grid interoperability is carried out within the framework of the GIN initiative [10] of the OGF. As written there: “The purpose of this group is to organize and manage a set of interoperation efforts among production grid projects interested in interoperating in support of applications that require resources in multiple grids.” The GIN related web page of the UK NGS [11] writes: “Grid Interoperation Now (GIN) is a Community Group of the Open Grid Forum (OGF). It aims to organize and manage a set of interoperation efforts among production grid projects to support applications that require resources in multiple grids.” Obviously the goal of the GIN is very close to the objectives of establishing a WWG although their ambitions do not go so far. The phase 1 tasks of the GIN VO is “to plan and implement interoperation in specific areas, initially data location and movement, authentication/authorization and identity management, job description and execution, and information services.”

The GIN has created the GIN VO with resources from the major production grids: TeraGrid [20], UK NGS [11], EGEE [21], OSG [22] and NorduGrid [23]. All these grids allocated some grid sites to do experiments on their interoperability. In the framework of the GIN VO activity a GIN Resource testing portal [12] has been set up based on the P-GRADE/GEMLCA portal technology [19]. This portal enables the job submission (even workflow submission) to all these grid sites and hence can be used to constantly monitor their availability and usability in the GIN VO. Although the goals of the GIN and the WWG described in this paper have many similarities the concept of the GIN is quite different from the implementation concept of the WWG.

A major European effort in providing grid interoperability between gLite, UNICORE and Globus is the OMII-Europe project [13] that tries to establish interoperability at the level of five services (JSDL, OGSA-DAI, VOMS, RUS and GridSphere).

The World Wide Grid testbed [14] initiated by Monash University has the same name as we used in this paper but their WWG has a quite different meaning than our WWG. Their WWG is not about connecting the existing production grids in order to establish an interoperable WWG, rather it is a volunteer grid test-bed specifically intended to test the grid economy concept developed in the Grid Economy project [15]. Their recent paper on InterGrid [16] shows many similarities with our concept of connecting existing grids into a WWG. They introduce InterGrid Gateways to connect the different grids while we propose the usage of meta-brokers. They do not emphasize the importance of advance grid portals and workflow interoperability but they put more emphasis on grid economy.

The meta-broker concept for providing grid interoperability is quite new and was first proposed by SZTAKI [17] at the CoreGrid workshop organized in conjunction with EuroPar'2006. Since that time another CoreGrid partner, the Technical University of Catalonia (UPC) has started to work on this concept. The redefinition of the Broker Property Description Language (BPDL) [9] is an on-going joint work between SZTAKI and UPC. The detailed architecture plan of the meta-broker is described in [5].

6 Conclusions

We have seen that the highest level of parallelism can be exploited in the WWG if workflows are executed as parameter sweep applications. In order to exploit the largest possible parallelism the most advanced architecture concept of the WWG is based on the BGBR model. In the BGBR model every client can be connected in a standard way to one of the uniform meta-brokers. As a result to build the WWG, the only thing we have to do is to define and implement:

1. The functionality of the meta-brokers
2. The intercommunication protocol of meta-brokers
3. The communication protocol of clients and meta-brokers
4. The standard intercommunication protocol of meta-brokers and grid brokers

Once these requirements are defined and implemented, any existing grid can be connected to the WWG provided that the broker of the given grid realizes the standard intercommunication protocol of meta-brokers and grid brokers. Even if requirement 4 is not fulfilled, those grids for which the Translator and Invoker module of the

meta-broker is already available can be connected to the WWG. The meta-broker should be implemented as an open source grid service in order that any grid could extend it with the necessary Translator and Invoker module by which the given grid could be accessed by the Meta-Broker.

Overall, we can state that there is no real technical obstacle to create a scientific computational World Wide Grid where complex parameter sweep workflow applications could run and exploit the largest possible parallelism. It means that technically the WWG can be established by simply introducing the meta-broker concept and using a network of uniform meta-brokers to connect the existing production grids. Obviously, the meta-broker itself does not help in managing workflows, only to submit nodes (jobs or service requests) of the workflow in a grid transparent way. To assist the workflow execution in the WWG workflow managers are needed. Moreover these workflow managers also should be interoperable. This issue has been investigated in a CoreGrid technical report [18].

Acknowledgements

The research described in this paper was carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265) and particularly in the work of two institutes of the CoreGrid project: Resource Management and Scheduling Institute and Grid Systems, Tools and Environments Institute.

References

1. Woogle web service search engine, http://data.cs.washington.edu/webService/
2. SAL, http://www.salcentral.com/salnet/srchsal.htm
3. eSigma's web service search engine, http://www.esigma.com/telos/discover/browseservices.html?cid=0,telos:categories:general:utilities
4. Kertesz, A., Rodero, I., Guim, F.: Data Model for Describing Grid Resource Broker Capabilities. In: CoreGRID Workshop on Grid Middleware in conjunction with ISC 2007 conference, Dresden, Germany, June 25-26 (2007)
5. Kertesz, A., Kacsuk, P.: Meta-Broker for Future Generation Grids: A new approach for a high-level interoperable resource management. In: CoreGRID Workshop on Grid Middleware in conjunction with ISC 2007 conference, Dresden, Germany, June 25-26 (2007)
6. Anjomshoaa, A., et al.: Job Submission Description Language (JSDL) Specification, Version 1.0: http://www.gridforum.org/documents/GFD.56.pdf
7. GT 2.4: The Globus Resource Specification Language RSL v1.0: http://www.globus.org/toolkit/docs/2.4/gram/rsl_spec1.html
8. JDL Job description language, http://www.grid.org.tr/servisler/dokumanlar/DataGrid-JDL-HowTo.pdf
9. Kertesz, A., Rodero, I., Guim, F.: Meta-Brokering requirements and research directions in state-of-the-art Grid Resource Management, Technical report, TR-0116, Institute on Resource Management and Scheduling, CoreGRID - Network of Excellence (November 2007)

10. GIN Project, https://forge.gridforum.org/projects/gin
11. UK NGS web page on GIN, http://wiki.ngs.ac.uk/index.php?title=GIN
12. GIN Resource testing portal, https://gin-portal.cpc.wmin.ac.uk:8080/gridsphere/gridsphere
13. OMII-Europe Project, http://www.omii-europe.com/
14. World Wide Grid testbed, http://www.gridbus.org/ecogrid/wwg/
15. Grid Economy project, http://www.gridbus.org/ecogrid/
16. Assuncao, M.D., Buyya, R., Venugopal, S.: InterGrid: A Case for Internetworking Islands of Grids. In: Concurrency and Computation: Practice and Experience (CCPE). Wiley Press, New York (2007)
17. Kertesz, A., Kacsuk, P.: Grid Meta-Broker Architecture: Towards an Interoperable Grid Resource Brokering Service. In: CoreGRID Workshop on Grid Middleware in conjunction with Euro-Par 2006, Dresden, Germany, August 28-29, 2006, pp. 112–116. Springer, Heidelberg (2006)
18. Kacsuk, P., Kiss, T.: Towards a scientific workflow-oriented computational World Wide Grid, CoreGrid Technical Report, TR-0115, Institute on Grid Systems, Tools and Environments, CoreGRID - Network of Excellence (December 2007)
19. Kacsuk, P., Sipos, G.: Multi-Grid, Multi-User Workflows in the P-GRADE Grid Portal. Journal of Grid Computing 3(3-4), 221–238 (2005)
20. The TeraGrid website, http://www.teragrid.org/
21. The gLite website, http://glite.web.cern.ch/glite/
22. The OSG website, http://www.opensciencegrid.org/
23. The NorduGrid website, http://www.nordugrid.org/

A List Scheduling Algorithm for Scheduling Multi-user Jobs on Clusters

J. Barbosa[1] and A.P. Monteiro[1,2]

[1] Universidade do Porto, Faculdade de Engenharia,
Departamento de Engenharia Informática
[2] INEB - Instituto de Engenharia Biomédica, Lab. Sinal e Imagem
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
{jbarbosa,apm}@fe.up.pt

Abstract. This paper addresses the problem of scheduling multi-user jobs on clusters, both homogeneous and heterogeneous. A user job is composed by a set of dependent tasks and it is described by a direct acyclic graph (DAG). The aim is to maximize the resource usage by allowing a floating mapping of processors to a given job, instead of the common mapping approach that assigns a fixed set of processors to a user for a period of time. The simulation results show a better cluster usage. The scheduling algorithm minimizes the total length of the schedule (*makespan*) of a given set of parallel jobs, whose priorities are represented in a DAG. The algorithm is presented as producing static schedules although it can be adapted to a dynamic behavior as discussed in the paper.

Keywords: Static and dynamic scheduling, parallel task, list scheduling, cluster computing.

1 Introduction

The aim of the work herein presented is to improve the performance of clusters in the processing of applications (or jobs) composed by a set of dependent tasks. The common scheduling approach is to consider a fixed number of available processors to schedule the set of tasks [13,14,17,18,20] which on a multi-user environment corresponds to fix the number of processors available for each user. The presented model is based on a former model [2] to schedule DAGs of dependent parallel tasks. A parallel task, also called malleable task, is a task that can be executed on any number of processors with its execution time being a function of the processors alloted to it [7,12,15].

The target computer platform is a cluster, either homogeneous or heterogeneous, with a dedicated network. Such clusters can be private clusters of some organization but also they can be the nodes of a Grid infrastructure which to have a good performance requires, at least, that the end clusters have also a good performance.

The cluster schedulers usually allow that users specify the number of processors required to run their jobs, which imposes a static allocation of the cluster

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 123–136, 2008.

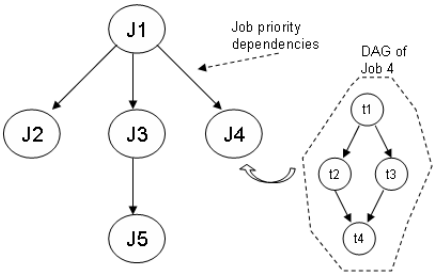


Fig. 1. A DAG composed by jobs of different users where job dependencies are created by priority policies; each job is described by a DAG of dependent tasks

nodes and a non-unified cluster management based only on user requests. Users try to allocate as much capacity as possible and there is not a global management. The proposed algorithm intends to optimize the cluster utilization by allowing different number of processors to be used along the processing of the tasks of a given user job. The algorithm is non-preemptive and achieves the goal by considering different number of processors to process the tasks of a given job. The DAG to schedule has two levels of detail. There is a master DAG, that establishes priorities among user jobs, and there is a DAG for each job. Figure 1 exemplifies a typical DAG. The scheduler input is the global DAG and all ready tasks are considered to schedule at a given time.

The common approach to schedule DAGs is the task parallel paradigm, which assigns one task to one processor. The scheduling consists on the distribution of the DAG nodes among the machine nodes, so that the *makespan* is minimum [1,11,18,19,20]. Here it is considered the parallel task model where each task can execute in more than one processor but one processor only participates in the execution of a task at any given time [7,12,21].

The remaining of the paper is organized as follows: section 2 defines the scheduling problem and revises related work in scheduling parallel and non-parallel tasks. Section 3 presents the computational model and the methodology used in this paper. Section 4 presents the list scheduling algorithm proposed in this paper. Finally, section 5 presents results and section 6 conclusions.

2 Problem Definition and Related Work

The problem addressed in this paper is the scheduling of a parallel application represented by a directed acyclic graph (DAG) on a distributed memory computer (i.e. a cluster). A DAG $G = (V, E)$, where V is the set of v nodes of the graph, representing the tasks to be processed, and E is the set of e edges, representing precedence among tasks and communication costs. For each node v_i it is defined a schedule start-time ($ST(v_i)$) and a finish-time ($FT(v_i)$), being the schedule length given by $max_i\{FT(v_i)\}$. Therefore, the goal of scheduling is to minimize $max_i\{FT(v_i)\}$ [13].

The above definition is valid either for homogeneous or heterogeneous machines and either for parallel tasks (executed on several processors) and non-parallel tasks (executed on one processor). The existing work on scheduling parallel tasks deals almost exclusively on homogeneous computers, and either dependent or independent tasks. The problem is known as NP-Complete so that several authors proposed polynomial approximation schemes [6,7,12,15,16,21].

The problem studied here considers the scheduling of general task dependency graphs and both homogeneous and heterogeneous clusters. Tasks are considered parallel and non-monotonic, this is, the execution time of task i, $t_{i,p}$, is considered to be non-monotonic so that there is a number p of processors for which $t_{i,p} < t_{i,p-1}$ and $t_{i,p} < t_{i,p+1}$. Mainly for heterogeneous clusters connected by a standard network it was shown that, due to communication constraints and task granularity, leaving processors in the idle state can reduce the processing time [3,4,5]. The solution proposed is based on the list scheduling technique used for non-parallel tasks [13,18,19,20].

DAG scheduling is commonly addressed as a non-parallel task problem [13,14,17,18,20], therefore the algorithm proposed in this paper is compared to the Heterogeneous Earliest-Finish-Time (HEFT) algorithm [20]. The authors compared several scheduling algorithms for heterogeneous computing and conclude that HEFT is the best one for scheduling DAGs on those systems. HEFT comprises two phases: first, there is a task prioritizing phase and second, a processor selection phase that selects the processor that minimizes the task finish time. It implements an insertion based policy which considers the possibility of inserting a task in an earliest idle time slot between two already scheduled tasks on a processor. The aim of comparing to HEFT is to show that the parallel-task approach can improve significantly the performance of a cluster, heterogeneous or not, in scheduling DAGs, with a scheduling algorithm of the same time complexity as HEFT, which is $O(v^2 \times P)$ for a DAG of v tasks and a P processor machine.

The former techniques are all static approaches of the mapping problem that assume static conditions for a given period of time. A dynamic approach intends to be more flexible concerning the availability of information about tasks arrival time and machine availability. Dynamic mapping of tasks is usually addressed as an independent task scheduling [9] problem. This approach can be applied here at the job level because these are independent and our master DAG is also based on job priorities and job deadlines. The dynamic scheduling can be applied with our scheduling algorithm in the following way: a dynamic policy like [9] specifies the DAG for a given scheduling instant and our algorithm scheduled tasks based on that DAG. The DAG is updated when new jobs arrive and a schedule instant happens when there are tasks ready to schedule. However, in this paper dynamic scheduling is not considered, because it requires more research and also it would obfuscate the main comparison that is to show that a parallel task scheduling achieves better performance than a non-parallel one on a cluster.

3 Computational Model

The computational platform considered is a distributed memory machine composed by P processors of possibly different processing capacities (heterogeneous cluster), connected by a switched network. It supports simultaneous communications between different pairs of machines. It is assumed that the application is represented by a DAG and the execution time of the tasks can be estimated at compile time or before starting the execution. The communications required to complete a task are included in the computation time as a function of the processors p used by that task. The inter-task communication is defined as a function of the computational time of the sender task and it is represented by the edges weight in the DAG.

The computational model that supports the estimation of the processing time, for each task, is based on the processing capacity S_i of processor i ($i \in [1, P]$) measured in $Mflop/s$, the network latency T_L, and the bandwidth ω measured in $Mbit/s$. The total computation time is obtained by summing the time spent communicating, T_{comm}, and the time spent in parallel operations, $T_{parallel}$. The time required to transmit a message of b elements is $T_{comm} = T_L + b\omega^{-1}$. The time required to compute the pure parallel part of the code, without any sequential part or synchronization time, on p processors is $T_{parallel} = f(n)/\sum_{i=1}^{p} S_i$. The numerator $f(n)$ is the cost function of the algorithm, measured in floating point operations, depending on problem size n.

As an example, for a matrix multiplication of (n, n) matrices, using the algorithm described in [10], the number of floating point operations is estimated to be $f(n) = 2n^3$. The total amount of data required to be transmitted in order to complete the algorithm on a grid of processors $P = r \times c$ is $n^2(r-1)$ across rows of processors and $n^2(c-1)$ across columns of processors, resulting in the total of $n^2(r+c-2)$ data elements. If the broadcast over a column or a row of processors is considered sequential, then they are transformed in $(r-1)$ and $(c-1)$ messages, respectively.

Finally, the time function for the matrix multiplication algorithm is given by:

$$T = T_{comm} + T_{parallel} = \frac{n^2(r+c-2)}{w} + T_L + \frac{2n^3}{\sum_{i=1}^{p} S_i} \tag{1}$$

This expression is computed for $p = 1$ to P and the number of processors that minimize the processing time is determined. The computation of the best processor grid for linear algebra kernels, on a heterogeneous machine, was discussed in [4].

4 Scheduling Algorithm

The scheduling algorithm is divided in two steps: first, a construction of a master DAG where each node is a user job and each edge represents a priority of one job over another, as shown in Figure 1; and second, a list scheduling algorithm, based on [2], that schedules the master DAG, this is tasks of all jobs in a unique DAG.

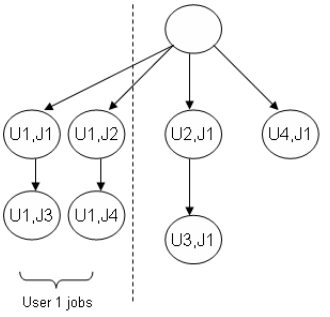


Fig. 2. Master DAG example; tasks of user 1 are organized separately to guarantee the reservation policy

The master DAG is created based on job priorities and deadlines [9]. Here it will be assumed that the master DAG is already defined and available to be scheduled (first step). In the second step the algorithm ensures that user reservation policy is not compromised such that, for example, if a user has reserved 20% of the cluster, their jobs will be schedule accordingly. The difference for the fixed capacity schedule is that if the user does not need that capacity at a given time, it will be available for other users. Figure 2 exemplifies the master DAG construction. If user 1 has reserved part of the machine, his tasks are put in parallel, at the top level. The tasks of other users are organized either in parallel or sequentially according to the prioritizing policy [9].

The master DAG have artificial nodes in order to impose priorities among jobs with zero processing time. Jobs are independent so that the edges have zero communication costs. Communications are only considered inside each job.

The scheduling algorithm applied to the global DAG is a list scheduling technique [13] which consists in the following steps: a) determine the available tasks to schedule, b) define a priority to them and c) until all tasks are scheduled, select the task with higher priority and assign it to the processor that allows the earliest start-time. For parallel tasks the last step selects not one processor but several processors that allow the earliest start-time [2]. Note that at this step we refer to tasks that result from all jobs.

Two frequently used attributes to define the tasks priorities are the *t-level* (top-level) and the *b-level* (bottom-level). The *t-level* of a node n_i is defined as the length of the longest path from an entry node to n_i (excluding n_i). The *b-level* of a node n_i is the length of the longest path from n_i to an exit node. The nodes along the DAG with higher *b-level* belong to the critical path.

The execution time $t_{i,p}$ is considered to be non-monotonic so that there is a number p of processors for which $t_{i,p} < t_{i,p-1}$ and $t_{i,p} < t_{i,p+1}$. Let $t^*_{i,p}$ be the minimum processing time of task i on the heterogeneous machine, which is achieved when the fastest p processors are used. Other combination of p processors will result in less computational capacity and consequently more processing time. The specific best processor layout should be a parameter of the tasks so that it can be considered in the optimal ($t^*_{i,p}$) processing time computation. From

this definition we can estimate a lower bound for the *makespan* which is the sum of the minimum processing time of the tasks on the critical path: $t_\infty = \sum_i t^*_{i,p}$, which is the time required to solve all tasks assuming an unbounded number of processors [21], and in this case it means that any task has the cluster fully available for it.

The expected *makespan* is higher because not all concurrent tasks can execute on the fastest processors which may change dynamically the critical path. Lower priority tasks, after being scheduled, can be transformed in critical path tasks if the capacity of the machine is lower than the required capacity to obtain $t^*_{i,p}$ for all concurrent tasks. Therefore, the algorithm [2] evaluates dynamically the *b-level* of the tasks being scheduled and makes scheduling corrections in order to reduce the maximum *b-level* of those tasks.

The processing capacity required to achieve $t^*_{i,p}$ for task i considers the fastest processors and is defined as $S^*_i = \sum_{j=1}^{p} S_j$. It is obvious that if slower processors are used, the capacity that achieves minimum time for task i is $S'_i < S^*_i$, resulting $t'_i > t^*_i$. Since more processors are required to obtain S^*_i they would imply more communications and consequently more processing time; therefore, the minimum processing time achievable will be certainly higher than the estimated t^*_i.

Algorithm1

1. `while tasks` $\neq \oslash$
2. `  Compute the set of ready tasks`
3. `  For each User k with limit>0`
3. `    For each ready task i`
4. `      Compute the optimal capacity` S^*_i
5. `    if` $\sum_i S^*_i > S_{limit}$
6. `      For each ready task i`
7. $S'_i = (S_{limit} / \sum_j S^*_j) S^*_i$
8. `      else`
9. `        For each ready task i` $S'_i = S^*_i$
10. $S_{max} = S_{max} - \sum_i S'_i$
11. `  For each ready task i`
12. `    Compute the optimal capacity` S^*_i
13. `  if` $\sum_i S^*_i > S_{max}$
14. `    For each ready task i`
15. $S'_i = (S_{max} / \sum_j S^*_j) S^*_i$
16. `  else`
17. `    For each ready task i` $S'_i = S^*_i$

Algorithm 1 is based on [2]. $S_{max} = \sum_{i=1}^{P} S_i$ is the total capacity of the homogeneous or heterogeneous machine computed as the sum of the individual capacity of each node. The while cycle at line 1 refers to all tasks of the DAG. In line 2 the ready tasks are those for which the predecessors have finished processing. From line 3 to 9 the algorithm determines the computational capacity that minimizes each ready tasks S^*_i for the users that have reserved a slice of the cluster

(represented by S_{limit}), according to the computational model and by assuming that the fastest processors are used. The number of processors is not important here and it is not registered. Then if the user limit S_{limit} is exceeded, the capacity assigned to each tasks is limited to the relative weight of each task. On line 10 the capacity left to other users that have no cluster reservation is computed. From line 11 to 17 the algorithm computes the same as from line 3 to 9 for the remaining tasks. The time complexity of algorithm 1 is $O(v \times P + v)$ since each task is only computed once: step 4 and 12 are $O(v \times P)$; and step 6 and 14 are $O(v)$.

Algorithm2

```
1.   Compute tl
2.   while ready tasks ≠ ∅
3.      Select the minimum tl
4.      Select processors that allow tl
5.      while  S_i < S'_i and t_{i,p} < t_{i,p-1}
           add processor
6.   Compute bl
7.   while true
8.      Select task k with highest bl
9.      Select task r with minimum bl
10.     if r has been maximum
           then break
11.     Reduce one processor to task r
12.     Assign it to task k
13.     Re-evaluate processing time
           of tasks r and k
14.     Re-evaluate bl of tasks r and k
```

Algorithm 2 is the second part of the scheduling algorithm. Here *t-level* and *b-level* were replaced by `tl` and `bl` respectively. From line 1 to 5 the algorithm schedules all ready tasks trying to assign to them the processing capacity determined before in Algorithm 1. The selected processors allow the tasks to start on their earliest start-time, but it also verifies if starting later, with more processors, they can finish at an earlier time. The processing time needs to be tested since in general the processors used are not the fastest ones and consequently the minimum processing time is achieved with less processing capacity, although higher than $t^*_{i,p}$.

From line 6 to 14 the algorithm tries to correct the last schedule by assigning more processors to the tasks that have higher *b-level*. The computation of *b-level* on line 6 and 14 uses $t^*_{i,p}$ for the tasks on following levels (not processed yet). For tasks of the current level, the time computed on line 5 and 13 are respectively used. The algorithm stops if the task with minimum *b-level* has been maximum during the minimization procedure. At line 11 the computation time of tasks r and k are re-evaluated considering the new set of processors assigned which have resulted from the transference of one processor from the set of r to the set of k.

The time complexity of *b-level* and *t-level* is $O(e+v)$ [13]. The time complexity of steps 3 to 5 is $O(v \times P)$ since it is executed once for each task and combined up to P processors. Steps 7 to 14 are $O(v^2 \times P)$ since each task can be compared to the others (v^2) and each of those cases is combined up to P processors. The resulting time complexity of both algorithms is $O(v^2 \times P)$.

The algorithm can be applied at the beginning of the computation and generates all the scheduling, resulting in a static scheduling. But if Algorithm 1 and 2 are executed every time that new ready tasks are available, this is, with the feedback of the computation and eventually with new jobs that may have arrived, it will produce a dynamic scheduling that takes into account the availability of the nodes (some may go off), the new jobs submitted and the expiration of user reservations. In fact for a cluster only a dynamic behavior will be useful.

5 Results and Discussion

In this section the evaluation of the scheduling algorithm proposed in this paper and a comparison to the HEFT [20], a reference algorithm of the related work section, is presented. The results shown below are obtained from a simulation setup but based on measures taken in the target cluster. The procedure to estimate computation and communication times were presented and analyzed before [4].

5.1 Parallel Machine

Although both scheduling algorithms were designed to work on heterogeneous machines, the machine considered here is homogeneous in order to have an unbiased comparison of the algorithms behavior. In fact a homogeneous computer is a particular case of the general heterogeneous paradigm.

The machine considered is composed by 20 processors, connected by a $100Mbit$ switched Ethernet. The processors are Pentium IV at 3.2 GHz and 1 GB of RAM with an individual capacity of $404Mflops$. The main characteristic of the network is that it allows simultaneous communications between different pairs of machines. For parallel tasks this is an important characteristic because to complete a task the involved processors (group) have to exchange data. In the general case, when accounting for the amount of communication involved, we need to ensure that inside the group there is no communication conflicts. Otherwise, it would be very difficult to synchronize communications, in different parallel tasks, to avoid conflicts.

5.2 DAGs and Tasks

There were used three DAGs, one with 10 tasks obtained from [20] and shown in Figure 3, for direct comparison, and two other DAGs of 30 and 90 tasks. These last two DAGs were generated based on the algorithm presented in [8] which can be resumed as follows: there are Na nodes with no predecessors and only successors, with ids ranging from 1 to Na; Nb nodes with both predecessors and successors, with ids ranging from Na+1 to Na+Nb; Nc nodes with only

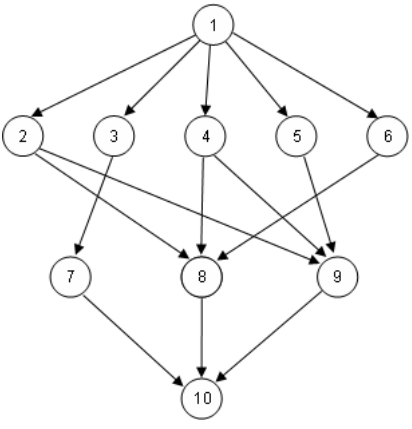


Fig. 3. Sample DAG with 10 nodes obtained from [20]

predecessors and ids ranging from Na+Nb+1 to Na+Nb+Nc. Here we considered Na=Nc=4 and Nb equal to the remaining nodes. The minimum and maximum out node degree is 2 and 5 respectively. We also make all edges pointing from smaller id nodes to larger id nodes.

The structure of the randomly generated DAG [8] can represent a collection of jobs from one or several users that are organized in a master DAG as expressed on section 4, and representing real applications. In this paper the tasks that form all DAGs are linear algebra kernels namely tridiagonal factorization (TRD), matrix Q computation, QR iteration and correlation (C). The size of each task is randomly generated in the interval $[100, 400]$. The processing times estimated in the scheduling algorithm are based on real values measured on the target processors. Table 1 shows the relative computation and intra-task communication weight of the tasks. The DAG edges are assigned a inter-task communication cost of 30% of the computation time of the precedent task (computation to communication ratio of 0.3).

5.3 Limitations of the Non-parallel Task Scheduling

First, it is shown that a non-parallel approach does not take advantage of the capacity available mainly due to the serialization of tasks in the same processor in order to reduce inter-task communications. The scheduling resulted from

Table 1. Relative computational and intra-communication weights of tasks

Task type	TRD	Q	QR	C
Task relative computational weight	1	0.82	2	3
Task relative communication weight	1	0.125	0.25	0.50

Table 2. Schedule length obtained with HEFT and Parallel Task scheduling

Algorithm	DAG10	DAG30	DAG90
HEFT (s)	7.87	23.85	27.64
Parallel task scheduling (s)	4.02	14.31	21.41

these algorithms, like HEFT, uses few processors because this results in an optimization of the scheduling length. Figure 4-a) shows the computational load for a computer with 20 processors. It can be seen that for the 10 node DAG, of Figure 3, only 3 processors have significant load; for the 30 node DAG only 4 processors and with very different loads; and for the 90 node DAG, 5 processors are idle and other 5 have very low load, this is, half of the machine is idle almost of the time. This behavior shows that if a user reserves a set of nodes for a given period of time, it is not guaranteed that the machine is well used even if the user has heavy DAGs to execute.

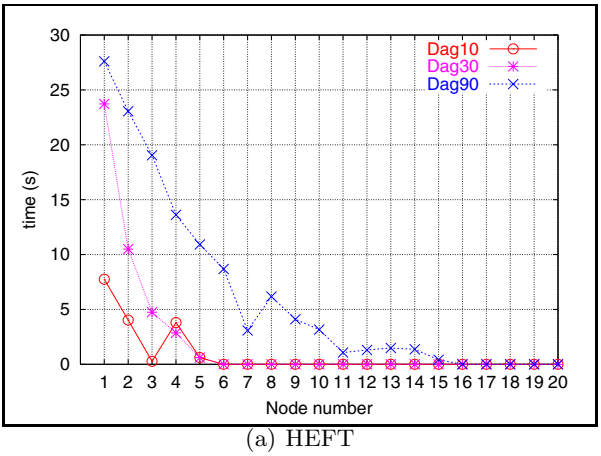
A parallel task scheduling, on the other hand, achieves better load distribution and consequently better machine usage. Figure 4-b) shows the load distribution for the same DAGs. Although this approach requires data redistribution between tasks and intra-task communications, it reduces the scheduling length as shown in table 2.

5.4 Scheduling with User Reservation of Machine Nodes

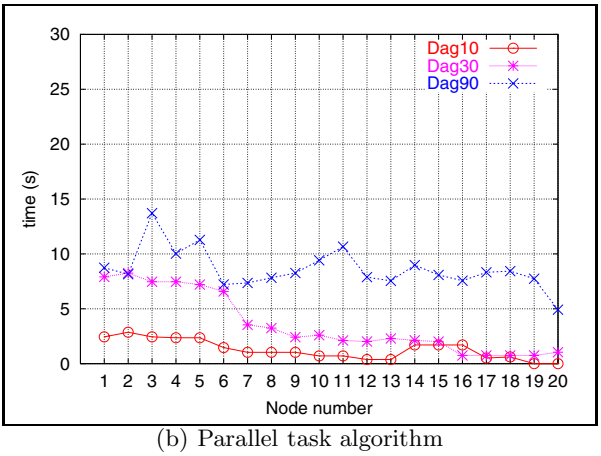
In point 5.3 it was shown the advantages of the parallel task scheduling. Figure 4 shows a better load balance for this algorithm but not a perfect one. A perfect scheduling would result in the same load for all processors assigned to a user. This may not be achievable due to DAG restrictions. Therefore, the alternative proposed with algorithms 1 and 2 is to make a flexible management of the nodes assigned to a user so that if at any given point the user cannot use that processing capacity, it will be available for jobs of other users. Instead of assigning processors it is assigned processing capacity so that along a DAG execution different processors can be used with the restriction of having, together, the same processing capacity. This strategy is straightforward applied for the heterogeneous case.

We distinguish two situations that are: a) the user tasks are independent or the DAG executed allows an efficient usage of the processors reserved with few idle periods; and b) the user DAG imposes some idle periods in the processors reserved. In the first situation, the algorithm proposed here does not bring any advantage, apart from the task parallel scheduling that may improve the schedule as shown in point 5.3. It is for the second hypothesis that the usage of the machine as a whole can be improved and consequently the schedule length of the jobs in general. Next, it will be shown with a situation example how the scheduling algorithm improves the global performance.

Consider that a user reserves 12 nodes of a 20 node computer for a period of time. However, user1 runs a job with 10 nodes that finishes before the



(a) HEFT



(b) Parallel task algorithm

Fig. 4. Load distribution obtained with HEFT and parallel task scheduling algorithm

reserved period. Another user wants to run a job with 30 nodes which would use the remaining 8 nodes. Table 3 resumes the schedule length obtained when an exclusive usage is imposed and when a flexible usage is allowed with the global management as proposed here.

In this case even user1 that consumed less computing power than the one reserved was able to reduce the schedule length of the job. This is due to the utilization of 13 processors in one given moment of the processing. This happened due to rounding effects that resulted in the assignment of one more processor to user1. The algorithm assigns computing power, but in shanks equivalent to the computing power of the powerful node available. What was expected was to obtain the same processing time as in the case of exclusive usage. The other user reduces substantially the schedule length because when user1 does not use the reserved power it is assigned to the other job. Figure 5 shows the assignment of nodes to the tasks of user1 and user2 jobs when using the global management. It

Table 3. Schedule length obtained with exclusive use of nodes and a global management

user	Schedule length (s)	
	Exclusive use	Global Management
User1 (12 nodes)	106.59	93.60
User2 (8 nodes)	251.78	210.44

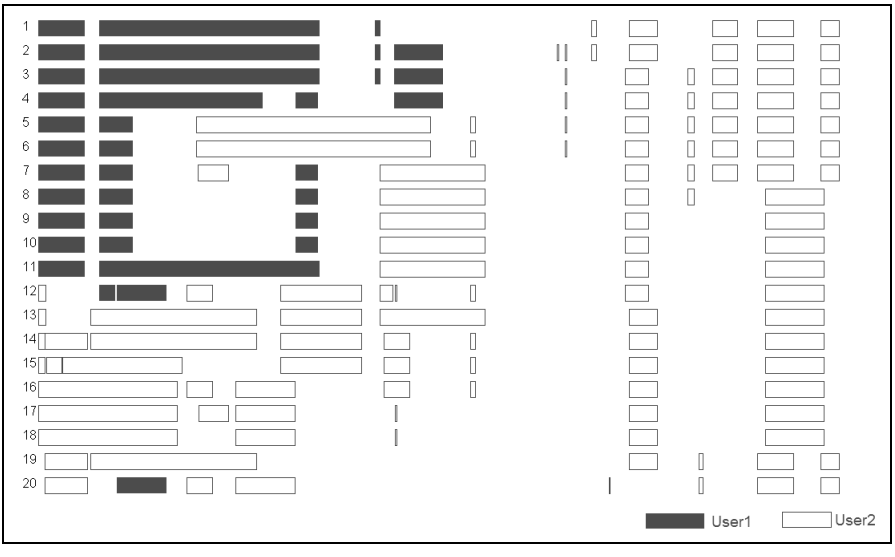


Fig. 5. Mapping of tasks to computer nodes with the global management; node number in the vertical axis and time in the horizontal axis

can be seen that one task of user1 is executed on processor number 20 and that before the end of the user1 job, user2's job uses processors in the set of the first 12. This is because task restrictions in user1 DAG left several nodes idle. The gaps between tasks of a given user are due to inter-task communications. The gap is proportional to the edges arriving a node, because the algorithm sums those communications.

6 Conclusions

The scheduling algorithm presented in this paper can improve the cluster utilization and the response time once we allow a variable computing power (number of processors) assigned for a job. Although the results presented are for a homogeneous cluster, the algorithm was designed for heterogeneous machines. To overcame the heterogeneity of the machine, the algorithm starts by computing the amount of capacity in $Mflops$, instead of number of processors, that minimizes the processing time of each task. For that it uses the fastest processors

and determines the minimum processing time that each task can achieve in that machine. Then, the algorithm joins processors until the maximum capacity required for each task is achieved, independently of the number of processors, but restricted by the maximum capacity available.

The algorithm proposed does not require a fixed subdivision of processors. When scheduling a set of ready tasks the machine is viewed has a whole, independently of the groups of processors formed in the last level, thus allowing a better use of the machine and consequently achieving improvements in processing time.

It was demonstrated that, when scheduling DAGs, a non-parallel task scheduling has limitations to efficiently use a set of processors assigned to a job.

Acknowledgments

The work presented was partially done in the scope of the project Segmentation, Tracking and Motion Analysis of Deformable (2D/3D) Objects using Physical Principles, with reference POSC/EEA-SRI/55386/2004, financially supported by FCT-Fundação para a Ciência e Tecnologia from Portugal.

References

1. Amoura, A.K., Bampis, E., König, J.-C.: Scheduling algorithms for parallel gaussian elimination with communication costs. IEEE Transactions on Parallel and Distributed Systems 9(7), 679–686 (1998)
2. Barbosa, J., Morais, C., Nobrega, R., Monteiro, A.P.: Static scheduling of dependent parallel tasks on heterogeneous clusters. In: Heteropar 2005, pp. 1–8. IEEE Computer Society, Los Alamitos (2005)
3. Barbosa, J., Padilha, A.J.: Algorithm-dependent method to determine the optimal number of computers in parallel virtual machines. In: Hernández, V., Palma, J.M.L.M., Dongarra, J. (eds.) VECPAR 1998. LNCS, vol. 1573, pp. 508–521. Springer, Heidelberg (1999)
4. Barbosa, J., Tavares, J., Padilha, A.J.: Linear algebra algorithms in a heterogeneous cluster of personal computers. In: Proceedings of 9th Heterogeneous Computing Workshop, pp. 147–159. IEEE CS Press, Los Alamitos (2000)
5. Berman, F., Wolski, R., Figueira, S., Schopf, J., Shao, G.: Application-level scheduling on distributed heterogeneous networks. In: Supercomputing 1996 (1996)
6. Blazewicz, J., Dell'Olmo, P., Drozdowski, M., Maczka, P.: Scheduling multiprocessor tasks on parallel processors with limited availability. European journal of Operational Research (149), 377–389 (2003)
7. Blazewicz, J., Machowiak, M., Weglarz, J., Kovalyov, M., Trystram, D.: Scheduling malleable tasks on parallel processors to minimize the makespan. Annals of Operations Research (129), 65–80 (2004)
8. Shivle, S., et al.: Mapping of subtasks with multiple versions in a heterogeneous ad hoc grid environment. In: Heteropar 2004. IEEE Computer Society, Los Alamitos (2004)
9. Kim, J.-K., et al.: Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment. Journal of Parallel and Distributed Computing 67, 154–169 (2007)

10. Geijn, R., Watts, J.: Summa: Scalable universal matrix multiplication algorithm. Technical Report CS-95-286, University of Tennessee, Knoxville (1995)
11. Gerasoulis, A., Yang, T.: On the granularity and clustering of directed acyclic task graphs. IEEE Transactions on Parallel and Distributed Systems, 686–701 (June 1993)
12. Jansen, K.: Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme. Algorithmica 39, 59–81 (2004)
13. Kwok, Y., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys 31(4), 406–471 (1999)
14. Kwok, Y., Ahmad, I.: On multiprocessor task scheduling using efficient state space search approaches. Journal of Parallel and Distributed Computing 65, 1515–1532 (2005)
15. Lepère, R., Mounié, G., Trystram, D.: An approximation algorithm for scheduling trees of malleable tasks. European journal of Operational Research (142), 242–249 (2002)
16. Oh-Heum, Chwa, K.-Y.: Scheduling parallel tasks with individual deadlines. Theoretical Computer Science 215, 209–223 (1999)
17. Park, G.-L.: Performance evaluation of a list scheduling algorithm in distributed memory multiprocessor systems. Future Generation Computer Systems (20), 249–256 (2004)
18. Shirazi, B., Wang, M., Pathak, G.: Analysis and evaluation of heuristic methods for static task scheduling. Journal of Parallel and Distributing Computing 10, 222–232 (1990)
19. Sinnen, O., Sousa, L.: List scheduling: extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures. Parallel Computing (30), 81–101 (2004)
20. Topcuoglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems 13(3), 260–274 (2002)
21. Trystram, D.: Scheduling parallel applications using malleable tasks on clusters. In: 15th International Conference on Parallel and Distributed Processing Symposium (2001)

Data Locality Aware Strategy for Two-Phase Collective I/O

Rosa Filgueira, David E. Singh, Juan C. Pichel, Florin Isaila, and Jesús Carretero

Universidad Carlos III de Madrid
Department of Computer Science
{rosaf,desingh,jcpichel,florin,jcarrete}@arcos.inf.uc3m.es

Abstract. This paper presents Locality-Aware Two-Phase (LATP) I/O, an optimization of the Two-Phase collective I/O technique from ROMIO, the most popular MPI-IO implementation. In order to increase the locality of the file accesses, LATP employs the Linear Assignment Problem (LAP) for finding an optimal distribution of data to processes, an aspect that is not considered in the original technique. This assignment is based on the local data that each process stores and has as main purpose the reduction of the number of communication involved in the I/O collective operation and, therefore, the improvement of the global execution time. Compared with Two-Phase I/O, LATP I/O obtains important improvements in most of the considered scenarios.

1 Introduction

A large class of scientific applications operates on a high volume of data that needs to be persistently stored. Parallel file systems such as GPFS [15], PVFS [11] and Lustre [12] offer scalable solutions for concurrent and efficient access to storage. These parallel file systems are accessed by the parallel applications through interfaces such as *POSIX* or *MPI-IO*. This paper targets the optimization of the MPI-IO interface inside ROMIO, the most popular MPI-IO distribution.

Many parallel applications consist of alternating compute and I/O phases. During the I/O phase, the processes frequently access a common data set by issuing a large number of small non-contiguous I/O requests [19,20]. Usually These requests originate an important performance slowdown of the I/O subsystem. Collective I/O addresses this problem by merging small individual requests into larger global requests in order to optimize the network and disk performance. Depending on the place where the request merging occurs, one can identify two collective I/O methods. If the requests are merged at the I/O nodes the method is called *disk-directed I/O* [7,21]. If the merging occurs at intermediary nodes or at compute nodes the method is called *two-phase I/O* [3,2].

In this work we focus on the *Two-Phase I/O* technique, extended by Thakur and Choudhary in *ROMIO*[10]. Based on it we have developed and evaluated the *Locality-Aware Two-Phase I/O* (*LATP I/O*) technique in which file data access is dependent on the specific data distribution of each process. The comparison with the original version of Two-Phase I/O shows that our technique obtains an important run time reduction . This is achieved by increasing the locality of the first phase and, therefore, reducing the number of communication operations.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 137–149, 2008.

This paper is structured as follows. Section 2 contains the related work. Section 3 explains in detail the internal structure of *Two-Phase I/O*. Section 4 contains the description of the *Locality-Aware Two-Phase I/O*. Section 5 is dedicated to performance evaluations. Finally, in Section 6 we present the main conclusions derived from this work.

2 Related Work

There are several collective I/O implementations, based on the assumption that several processes access concurrently, interleaved and non-overlapping a file (a common case for parallel scientific applications). In disk-directed I/O [7], the compute nodes fordward file requests to the I/O nodes. The I/O nodes merge and sort the requests before sending them to disk. In server-directed I/O of Panda [21], the I/O nodes sort the requests on file offsets instead of disk addresses. *Two-Phase I/O* [3,2] consists of an access phase, in which compute nodes exchange data with the file system according to the file layout, and a shuffle phase, in which compute nodes redistribute the data among each other according to the access pattern. We present this implementation of theses technique in ROMIO in the next section. Using Lustre file joining (merging two files into one) for improving collective I/O is presented in [22].

Several researchers have contributed with optimizations of MPI-IO data operations: data sieving [10], non-contiguous access [16], collective caching [17], cooperating write-behind buffering [18], integrated collective I/O and cooperative caching [14].

3 Internal Structure of Two-Phase I/O

As its name suggests, *Two-Phase collective I/O* consists of two phases: a shuffle phase and an I/O phase. In the shuffle phase, small file requests merged into larger ones. In the second phase, contiguous transfers are performed to or from the file system.

Before these two phases, *Two-Phase I/O* divides the file region between the minimum and maximum file offsets of accesses of in equal contiguous regions,called *File Domains* (FD) and assigns each FD to a configurable subset of compute nodes, called aggregators. Each aggregator is responsible for aggregating all the data inside its assigned FD and for transferring the FD to or from the file system.

In the implementation of *Two-Phase I/O* the assignment of FD to aggregators is fixed, independent of distribution of data over the compute nodes. In contrast, based the processor data distribution, LATP minimises the total volume of communications. By means of this strategy it is possible to reduce the communication and, therefore, the overall I/O time.

We ilustrate the *Two-Phase I/O* technique through an example of a vector of 16 elements that is written in parallel by 4 processes (see Figure 1) to a file. The size of one element is 4. Each process has previously declared a view on the file, i.e. non-contiguous regions are “seen” as if they were contiguous. For instance, process 0 “sees” the data at file offsets 0-3 and 20-23 contiguously, as view offsets 0-7.

Before performing the two mentioned phases, each process analyzes, which parts of the file are stored locally by creating a list of offsets and list of lengths. According to the example, process 0 is assigned three intervals: (*offset*=0, *length*=4), (*offset*=20,

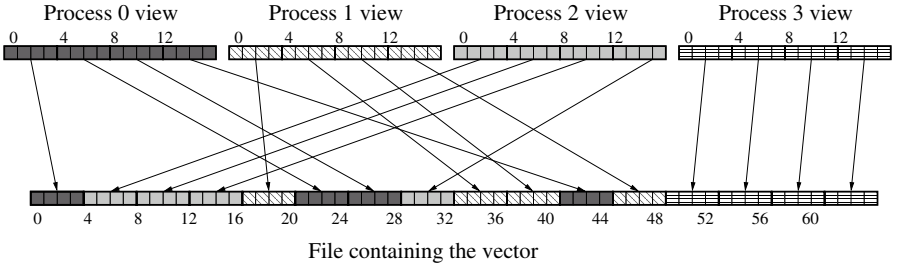


Fig. 1. File access example

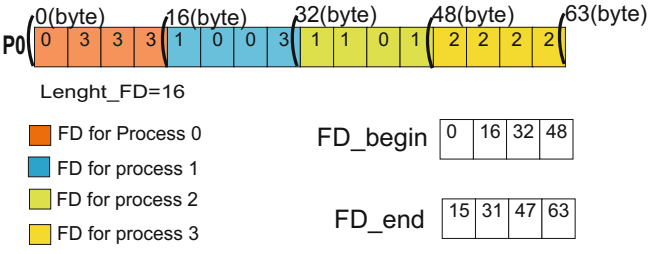


Fig. 2. Assignment of file domain

length=8), (*offset*=40, *length*=4). The list of offsets for this process is: {0, 20, 40} and the list of lengths is: {4, 8, 4}.

In addition, each process calculates the first and last byte of the accessed intervl. In our example, the first byte that process 0 has stored is 0 and the last byte is 43. Next, all processes exchange these values and compute the maximum and minimum of file access range, in this case 0 and 63, respectively. The interval is then divided in to equal-sized FDs. If all 4 compute nodes are aggregators, it will be divided in 4 chunks of 16 bytes, one for each aggregator. Each chunk is assigned to each aggregator according to its rank value. That is, block 0 is assigned to process with rank 0, block 1 to rank 1, etc. Each chunk (FD) is written to file by the assigned process. For performing this operation, each process needs all the data of its FD. If these data are stored in other processes, they are exchanged during the communication phase.

Once the size of each FD is obtained, two lists with so many positions as number of processes are created. The first list indicates the beginning of the FD of each process. The second one indicates the final of the FD of each process. Figure 2 shows how the vector is divided into different FDs. Each FD has been assigned a different color. Also, it can be observed that the assignment of FD is independent of the local data of each process. This scheme is inefficient in many situations. For example, the FD for process 3, begins at byte 48 and finalizes at byte 63. All these data are stored in the process 2, so this implies unnecessary communications between process 2 and 3, because the process 2 has to send all this data to process 3, insted of writing it to disk.

Once each process knows all the referring data, it analyzes, which data from its FD is not locally stored and what communication has be to be established in order to gather the data. This stage is reflected in Figure 3. This figure shows the data of the P vector

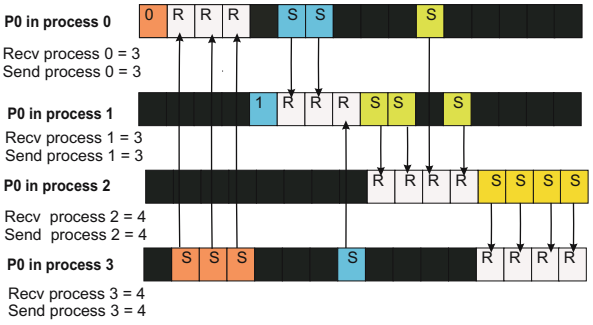


Fig. 3. Data transfers between processes

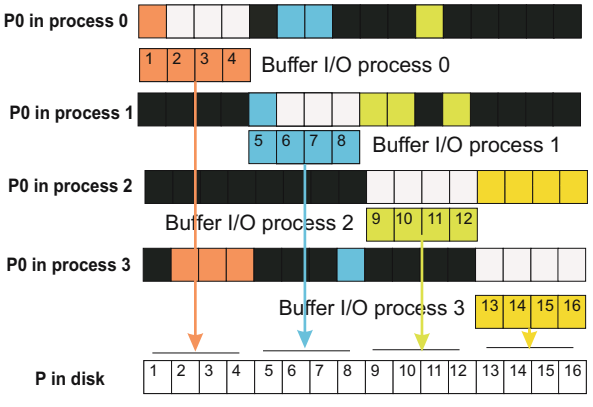


Fig. 4. Write of data in disk

that each process has locally stored. For any process, the vector elements labeled 'R' are received and the ones labled 'S' are sent. The arrows represent communication operations.

For example, in Figure 3, process 0 needs three elements that are stored in the process 3, and has stored three elements that processes 1 and 2 need. In the following step of *Two-Phase I/O* technique, the processes exchange the previously calculated data. Once all the processes have the data of their FD, they write to file a chunk of consecutive entries as shown in Figure 4. Each process transfers only one contiguous region to file (its FD), thus, reducing the number of I/O requests and improving the overall I/O performance.

4 Locality Aware Strategy for Two-Phase I/O

As explained in Section 3, *Two-Phase I/O* makes a fixed assignment of the FDs to processes. With the *LA-Two-Phase I/O* replaces the rigid assignment of the FDs by an assignment dependent of the initial distribution of the data over the processes. Our approach is based on the Linear Assignment Problem.

4.1 Linear Assignment Problem

The *Linear Assignment Problem* (LAP) is a well studied problem in linear programming and combinatorial optimization. LAP computes the optimal assigment of n items to n elements given an $n \times n$ cost matrix. In other words, LAP selects n elements of the matrix (for instance, the matrix from Table 1), so that there is exactly one element in each row and one in each column, and the sum of the corresponding costs is maximum.

The problem of finding the best interval assignment to the existing processes can be efficiently solved by applying the existing solutions of this problem. In our case, the LAP tries to assign FDs to processes, by maximizing the cost, given that we want to assign to the process the interval, for which it has more local data.

A large number of algorithms, sequential and parallel, have been developed for LAP. We have selected for our work the following algorithms, considered to be the most representative ones:

- *Hungarian algorithm* [1]: This is the first polynomial-time primal-dual algorithm that solves the assignment problem. The first version was invented and published by Harold Kuhn in 1955 and has a $O(n^4)$ complexity. This was revised by James Munkres in 1957, and has been known since as the Hungarian algorithm, the Munkres assignment algorithm, or the Kuhn-Munkres algorithm.
- *Jonker and Volgenant algorithm*[5]: They develop a shortest augmenting path algorithm for the linear assignment problem. It contains new initialization routines and a special implementation of Dijkstra's shortest path method. For both dense and sparse problems computational experiments they show this algorithm to be uniformly faster than the best algorithms from the literature. It has a $O(n^3)$ complexity.
- *APC and APS Algorithms*[4]: These codes implement the Lawler $O(n^3)$) version of the Hungarian algorithm by Carpenato, Martello and Toth. APC works on a complete cost matrix, while APS works on a sparse one.

4.2 LA-Two-Phase I/O

In order to explain *LA-Two-Phase I/O*, we use the example for Section 2 with the same data and distribution as in Figure 1.

The proposed technique differs from the original version in the assignment of the FDs to processes. Each FD is assigned based on the distribution of the local data of processes. In order to compute this initial distribution, the number of intervals in which we can divide the file is computed. This is made by dividing the size of the access interval by the sizes of the FD. In our example, the number of intervals is equal to four.

The next step consists in assigning each interval to each process efficiently. First, a matrix is constructed, with as many rows as processes, and so many columns as intervals. Each matrix entry contains the number of elements that each process has stored. The matrix from our example is shown in Table 1.

Our technique is based on maximizing the aggregator locality by applying a LAP algorithm and obtaining a list with the assignment of intervals to processes. For our

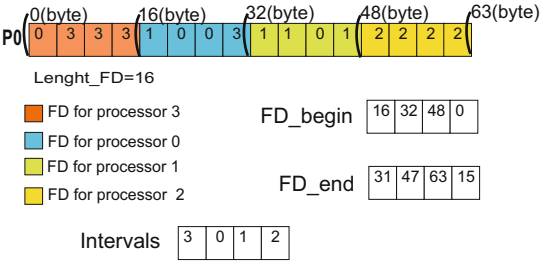


Fig. 5. Assignment of file domain for LA-Two-Phase I/O

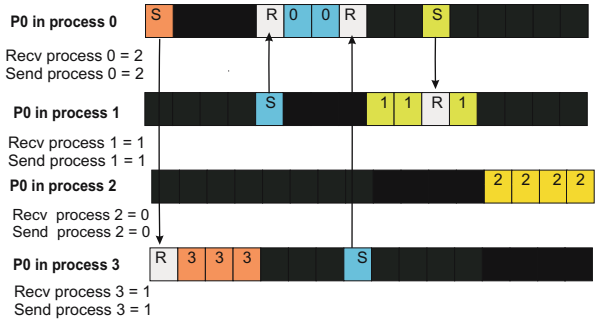


Fig. 6. Transfers of data between processes in LA-Two-Phase I/O

example, the assignment list is $\{3, 0, 1, 2\}$ as indicated Figure 5. This list represents the interval that has been assigned to each process.

This strategy reduces the number of communication operations, due to the fact that each process increases the amount of locally assigned data. The following phases of the *LA-Two-Phase I/O* are the same as those of the original version. Figure 6 shows the communication operations between processes. In the corresponding step of original *Two-phase I/O*, shown in Figure 3, process 2 sends four elements and receives four elements. With our technique, the number of transfers of process 2 has been reduced to none (see Figure 6). In this example the LA-Two-Phase I/O reduces the overall transfers from 28 exchanged elements to 8.

Figure 7 shows the I/O phase of *LA-Two-Phase I/O*. In the same way that as in the original technique, each process writes to file a contiguous data region.

Table 1. Number of elements of each interval

Interval/Process	0	1	2	3
0	1	2	1	0
1	0	1	3	0
2	0	0	0	4
3	3	1	0	0

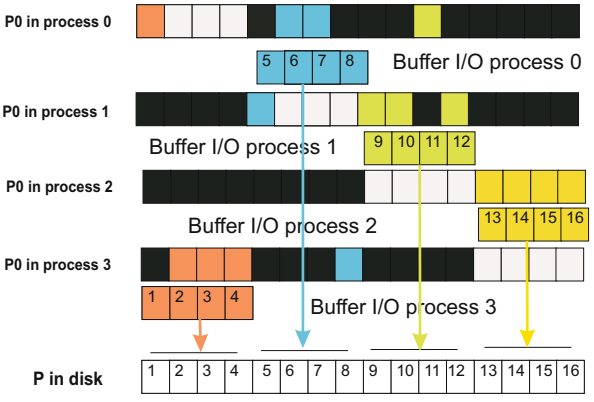


Fig. 7. Write of data in disk in LA-Two-Phase I/O

5 Performance Evaluation

The evaluations in this paper were performed by using the BIPS3D application with different input meshes related to different semiconductor devices. We have compared LATP I/O with the original version of the technique *Two-Phase I/O* implemented in MPICH.

The tests have been made in Magerit cluster, installed in the CESVIMA supercomputing center. Magerit has 1200eServer BladeCenter JS202400 nodes, and each node has two processors IBM 64 bits PowerPC single-core 970FX running at 2.2 GHz and having 4GB RAM and 40GB HD. The interconnection network is Myrinet.

We have used the MPICHGM 2.7.15NOGM distribution for the basic implementation of *Two-Phase I/O*. We have developed our technique by modifying this code. The parallel file system used is PVFS 1.6.3 with one metadata server and 8 I/O nodes with a striping factor of 64KB.

The remainder of this section is divided as follows. Subsection 5.1 briefly overviews the BIPS3D application. Subsection 5.2 contains the evaluation of the linear assignment technique. Finally, the evaluation of *LA-Two-Phase I/O* is presented in subsection 5.3.

5.1 BIPS3D Simulator

BIPS3D is a 3-dimensional simulator of BJT and HBT bipolar devices [8]. The goal of the 3D simulation is to relate electrical characteristics of the device with its physical and geometrical parameters. The basic equations to be solved are Poisson's equation and electron and hole continuity in a stationary state.

Finite element methods are applied in order to discretize the Poisson equation, hole and electron continuity equations by using tetrahedral elements. The result is an unstructured mesh. In this work, we have used four different meshes, as described later.

Using the METIS library, this mesh is divided into sub-domains, in such a manner that one sub-domain corresponds to one process. The next step is decoupling the

Table 2. Size in MB of each file based on the mesh and loads

Mesh/Load	$mesh1$	$mesh2$	$mesh3$	$mesh4$
100	18	12	28	110
200	36	25	56	221
500	90	63	140	552

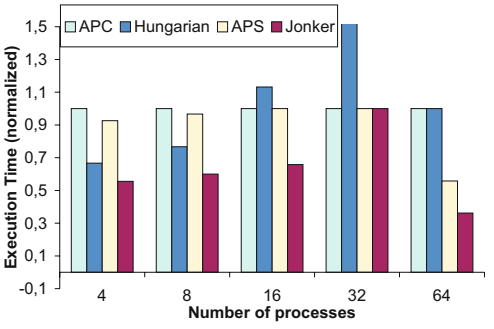


Fig. 8. Time for computing the optimal allocation for $mesh1$

Poisson equation from the hole and electron continuity equations. They are linearized by Newton method. Then we construct, for each sub-domain, in a parallel manner, the part corresponding to the associated linear system. Each system is solved using domain decomposition methods. Finally, the results are written to a file. For our evaluation BIPS3D has been executed using four different meshes: $mesh1$ (47200 nodes), $mesh2$ (32888 nodes), $mesh3$ (732563 nodes) and $mesh4$ (289648 nodes), with different number of processes: 8, 16, 32 and 64. The BIPS3D associates a data stucture to each node of a mesh. The contents of these data structures are the data written to disk during the I/O phase. The number of elements that this structure has per each mesh entry is given by the *load* parameter. This means that, given a mesh and a load, the number of data written is the product of the number of mesh elements and the load. In this work we have evaluated different loads, concretely, 100, 200 and 500. Table 2 lists the different sizes (in MB) of each file based on load and mesh characteristics.

5.2 Performance of the Linear Assignment Problem

We have applied all the LAP algorithms described in Section 4 to our problem. We have noticed that in all cases all algorithms produce the same assignment (of FDs to processes). The only difference between them is the time to compute the optimal allocation. Figure 8 shows the normalized execution time (taking the APC algorithm as the reference technique) for solving the interval distribution using different number of processes and $mesh1$ data distribution. Note that the fastest algorithm is the Jonker and Volgenant, and for this reason we have chosen it to apply in *LA-Two-Phase I/O*.

5.3 Performance Evaluation of LA-Two-Phase I/O

Figure 9 shows the percentage of reduction of communications for *LA-Two-Phase I/O* over *Two-Phase I/O* for $mesh1$, $mesh2$, $mesh3$ and $mesh4$ and different numbers of processes. We can see that, when LATP is applied, the volume of transferred data is considerably reduced.

In the first step of our study we have analyzed the *Two-Phase I/O*, identifying the stages of the technique that are more time-consuming. The stages of *Two-Phase I/O* are:

- *Offsets and lengths calculation (st1)*: In this stage the list of offsets and lengths of the file is calculated.
- *Offsets and lengths communication (st2)*: Each process communicates its start and end offsets to the other processes. In this way all processes have global information about the involved file interval.
- *Interval assignment (st3)*: This stage only exists in *LA-Two-Phase I/O*. First, it calculates the number of intervals into which we can divide the file, and then, it assigns intervals to processes by applying *Linear Assignment Problem* (see Table 1).
- *File domain calculation (st4)*: The I/O workload is divided among processes (see Figure 2). This is done by dividing the file into file domains (FDs). In this way, in the following stages, each aggregator collects and transfers to the file the data associated to its FD.
- *Access request calculation (st5)*: It calculates the access requests for the file domains of remote processes.
- *Metadata transfer (st6)*: Transfer the lists of offsets and lengths.
- *Buffer writing (st7)*: The data are sent to appropriate processes (see Figure 3).
- *File writing (st8)*: The data are written to file (see Figure 4).

The buffer and file writing stages (st7 and st8), are repeated as many times as the following calculus indicates: the size of the file portion of each process is divided by the size of the *Two-Phase I/O* buffer (4 MB in our experiments). First, the write size of each process is obtained by dividing the size of the file by the number of processes. For example, for $mesh4$ with load 500 and using 8 processes the size of the file is 552 MB (see table 2). Therefore, the write size of each process is 69 MB. Then, the file size

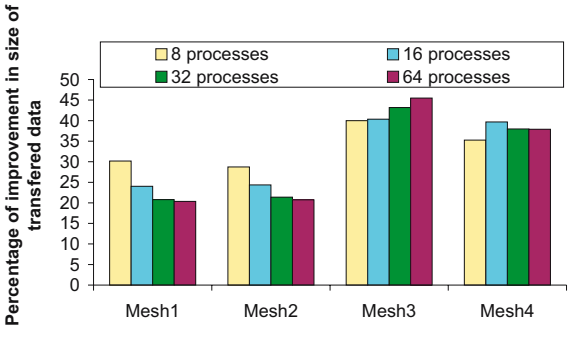
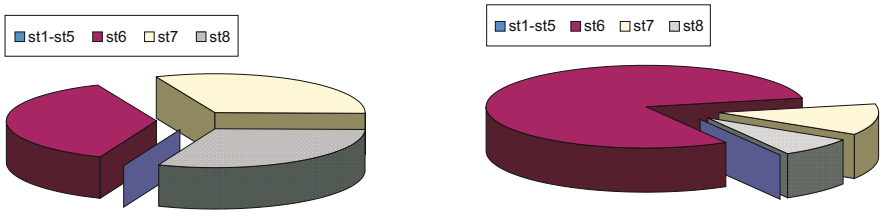
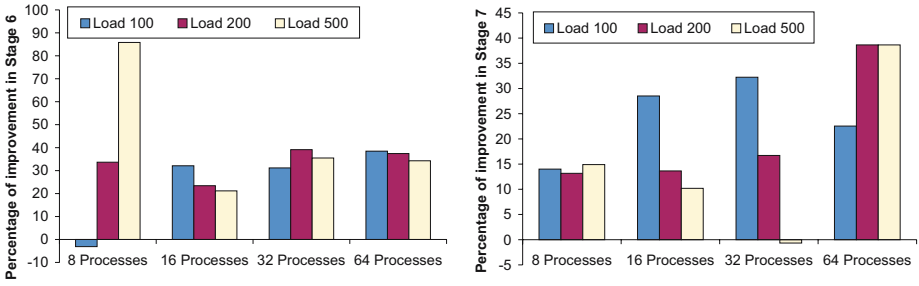


Fig. 9. Percentage reduction of transferred data volume for $mesh1$, $mesh2$, $mesh3$ and $mesh4$



(a) (b)

Fig. 10. Stages of Two-Phase I/O for $mesh1$: (a) with load 100 and 16 processes and (b) with load 100 and 64 processes



(a) (b)

Fig. 11. Percentage of improvement for $mesh1$: (a) in Stage 6 and (b) in Stage 7

related to each process is divided to the buffer size of *Two-Phase I/O*. Consequently, the number of times is given by this value divided by 4MB, for this example is 18.

Figure 11 represents the percentage of time of *Two-Phase I/O* for $mesh1$ with load 100, with 16 and 64 processes, respectively. The costs of stages st1, st2, st3, st4 and st5 have been added up, and we have represented this value in the figures as st1-st5.

As we can see in the Figures 10(a) and 10(b), the slowest stages are st6 and st7. Note that the cost of the st6 stage increases with the number of processes. These figures show the weight of the communication stages in the *Two-Phase I/O* technique. Moreover, we can see that the cost of these stages increases with the number of processors. Based on this, we conclude that this represents an important bottleneck in this technique. For this reason we have developed the *LA-Two-Phase I/O* technique with the aim of reducing the amount of communication. This technique reduces the global time of *Two-Phase I/O*.

LA-Two-Phase I/O technique improves the communication performance of st6 and st7 stages. Figure 11(a) shows the percentage of improvement in st6 stage for $mesh1$, for different loads and different number of processes.

In this figure we can see that the time of st6 stage is significantly reduced in most cases. In this stage each process calculates what requests of other processes lie in its file domain and creates a list of offset and lengths for each process, which has data stored

in its FD. Besides, in this stage, each process sends the offset and length lists to the rest of the processes. In *LA-Two-Phase I/O*, many of the data that each process has stored belong to its FD (given that data locality is increased) and therefore less offsets and lengths are communicated.

Figure 11(b) depicts the time of stage st7. Note that, again, this time is reduced in most of cases. This is because in this stage, each process sends the data that has calculated in st6 stage to the appropriate processes. In *LA-Two-Phase I/O*, many of the data that each process has stored belong to its FD, therefore, they send less data to the other processes, reducing the number of transfers and the volume of data.

Figure 12 shows the overall percentage of improvement of our technique for $mesh1$, $mesh2$, $mesh3$ and $mesh4$ with 64 processes. In this figure, we included the time of all stages. For this reason the percentage of improvement is smaller than in previous stages. Nevertheless, we can notice that in the majority cases a significant improvement in the execution time for *LA-Two-Phase I/O* technique. The original technique performed better in 4 of the 48 cases, but the loss was under 5% in all the cases. It appears that, for these cases (which represent less than 10% of the total), the data locality happened to be good enough in the original distribution and the additional cost to find a better distribution did not pay off.

It is important to emphasize that the additional cost of the new stage (st3) is very small compared with the total time. The fraction of this stage in the overall execution time is small: in the best case it is 0.07% of the time ($mesh2$, 8 processes and load 500) and in the worst case the 7% ($mesh3$, 64 processes and load 100).

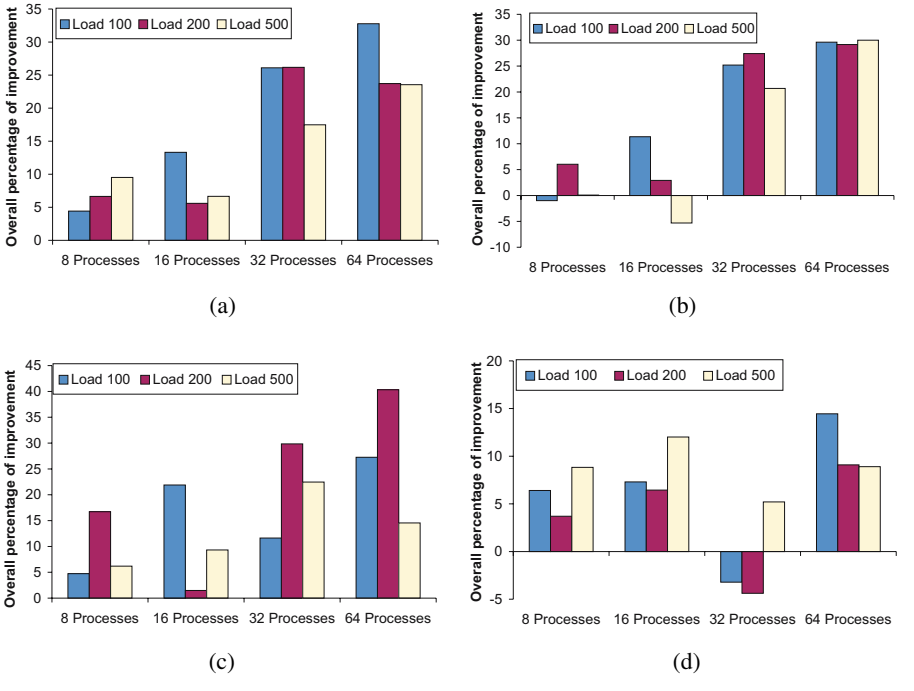


Fig. 12. Overall improvement for: (a) $mesh1$ (b) $mesh2$ (c) $mesh3$ and (d) $mesh4$

6 Conclusions

In this paper a new technique called *LA-Two-Phase I/O* based on the local data that each process stores is presented. First of all, we have showed that the *LA-Two-Phase I/O* improves the overall performance, when compared to the original *Two-Phase I/O*. The new stage (st3), which we have added to the technique *LA-Two-Phase I/O* has an insignificant overhead in comparison to the total execution time.

In the evaluation section we have shown that the greater number of processes, the larger the improvement brought by our technique. Finally, it is important to emphasize, that *LA-Two-Phase I/O* can be applied to every kind of data distribution.

Acknowledgements

This work has been partially funded by project TIN2007-63092 of Spanish Ministry of Education and project CCG07-UC3M/TIC-3277 of Madrid State Governement.

References

1. Blackman, S.S.: Multiple-Target Tracking with Radar Applications. Artech House, Dedham (1986)
2. Bordawekar, R.: Implementation of Collective I/O in the Intel Paragon Parallel File System: Initial Experiences. In: Proc. 11th International Conference on Supercomputing (July 1997) (to appear)
3. del Rosario, J., Bordawekar, R., Choudhary, A.: Improved parallel I/O via a two-phase run-time access strategy. In: Proc. of IPPS Workshop on Input/Output in Parallel Computer Systems (1993)
4. Giorgio Carpaneto, S.M., Toth, P.: Algorithms and codes for the assignment problem. Annals of Operations Research 13(1), 191–223 (1988)
5. Jonker, R., Volgenant, A.: A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems. Computing 38(4), 325–340 (1987)
6. Karypis, G., Kumar, V.: METIS — A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices. Technical report, Department of Computer Science/Army HPC Research Center, University of Minnesota, Minneapolis (1998)
7. Kotz, D.: Disk-directed I/O for MIMD Multiprocesses. In: Proc. of the First USENIX Symp. on Operating Systems Design and Implementation (1994)
8. Loureiro, A., González, J., Pena, T.F.: A parallel 3d semiconductor device simulator for gradual heterojunction bipolar transistors. Journal of Numerical Modelling: electronic networks, devices and fields 16, 53–66 (2003)
9. Seamons, K., Chen, Y., Jones, P., Jozwiak, J., Winslett, M.: Server-directed collective I/O in Panda. In: Proceedings of Supercomputing 1995 (1995)
10. Thakur, R., Gropp, W., Lusk, E.: Data Sieving and Collective I/O in ROMIO. In: Proc. of the 7th Symposium on the Frontiers of Massively Parallel Computation, pp. 182–189 (February 1999)
11. Ligon, W., Ross, R.: An Overview of the Parallel Virtual File System. In: Proceedings of the Extreme Linux Workshop (June 1999)

12. C.F.S. Inc. Lustre: A scalable, high-performance file system. Cluster File Systems Inc. white paper, version 1.0 (November 2002), http://www.lustre.org/
13. Indiana University, LAM website, http://www.lam-mpi.org/
14. Isaila, F., Malpohl, G., Olaru, V., Szeder, G., Tichy, W.: Integrating Collective I/O and Cooperative Caching into the "Clusterfile" Parallel File System. In: Proceedings of ACM International Conference on Supercomputing (ICS), pp. 315–324. ACM Press, New York (2004)
15. Schmuck, F., Haskin, R.: GPFS: A Shared-Disk File System for Large Computing Clusters. In: Proceedings of FAST (2002)
16. Thakur, R., Gropp, W., Lusk, E.: Optimizing Noncontiguous Accesses in MPI-IO. Parallel Computing 28(1), 83–105 (2002)
17. Liao, W.K., Coloma, K., Choudhary, A., Ward, L., Russel, E., Tideman, S.: Collective Caching: Application-Aware Client-Side File Caching. In: Proceedings of the 14th International Symposium on High Performance Distributed Computing (HPDC) (July 2005)
18. Keng Liao, W., Coloma, K., Choudhary, A.N., Ward, L.: Cooperative Write-Behind Data Buffering for MPI I/O. In: PVM/MPI, pp. 102–109 (2005)
19. Nieuwejaar, N., Kotz, D., Purakayastha, A., Ellis, C., Best, M.: File Access Characteristics of Parallel Scientific Workloads. IEEE Transactions on Parallel and Distributed Systems 7(10) (October 1996)
20. Simitici, H., Reed, D.: A Comparison of Logical and Physical Parallel I/O Patterns. In: International Journal of High Performance Computing Applications, special issue (I/O in Parallel Applications), vol. 12(3) (1998)
21. Seamons, K., Chen, Y., Jones, P., Jozwiak, J., Winslett, M.: Server-directed collective I/O in Panda. In: Proceedings of Supercomputing 1995 (1995)
22. Yu, W., Vetter, J., Canon, R.S., Jiang, S.: Exploiting Lustre File Joining for Effective Collective I/O. In: CCGRID 2007: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid, Washington, DC, USA, pp. 267–274. IEEE Computer Society, Los Alamitos (2007)

A Grid-Aware Web Portal with Advanced Service Trading for Linear Algebra Calculations

Hrachya Astsatryan[1], Vladimir Sahakyan[1], Yuri Shoukouryan[1], Michel Daydé[2], Aurelie Hurault[2], Marc Pantel[2], and Eddy Caron[3]

[1] Institute for Informatics and Automation Problems of the National Academy of Sciences of the Republic of Armenia, 1, P. Sevak str., Yerevan, 0014, Armenia
{hrach,svlad,shouk}@sci.am
[2] Institut de Recherche en Informatique de Toulouse, École Nationale Supérieure d´Électrotechnique, d´Électronique, d´Informatique, d´Hydraulique et des Télécommunications, 2, rue Charles Camichel, 31071, Toulouse, France
{Michel.Dayde,Aurelie.Hurault,Marc.Pantel}@enseeiht.fr
[3] Laboratoire de l'Informatique du Parallélisme
Université de Lyon - UMR ENS-Lyon, INRIA, CNRS, UCBL1 5668
46, Allée d'Italie, 69364, Lyon, France
Eddy.Caron@ens-lyon.fr

Abstract. Due to the rapid growth of the Internet, there has been a rising interest in using the Web as an interface to develop various applications over computational Grid environments. The purpose of this work is to develop a Grid-aware Web interface for linear algebra tasks with advanced service trading. Developing efficient and portable codes, requires users to face parallel computing and programming and to make use of different standard libraries, such as the BLAS [1], LAPACK [2] and ScaLAPACK [3] in order to solve computational tasks related to linear algebra. For this purpose, a scientific computing environment based on a Web interface is described that allows users to perform their linear algebra tasks without explicitly calling the above mentioned libraries and softwarep tools, as well as without installing any piece of software on local computers: users enter algebraic formula (such as in Matlab or Scilab [4]) that are evaluated for determining the combinations of services answering the user request. Services are then executed locally or over the Grid using the Distributed Interactive Engineering Toolbox (DIET) [5] middleware.

Keywords: Grid Computing, Linear Algebra, Scientific Computing, Web Portal, Service Trading.

1 Introduction

Scientific computing aims at constructing mathematical models and numerical solution techniques for solving problems arising in science (including life and social sciences) and engineering. The solution of linear system of equations lies at the heart of most calculations in scientific computing. For the past twenty years, there has been a great deal of activity in the area of algorithms and software for solving linear algebra problems. ScaLAPACK is a library of high-performance linear algebra

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 150–159, 2008.

routines for distributed-memory message-passing MIMD computers and networks of workstations supporting PVM and/or MPI. It contains routines for solving systems of linear equations, least squares problems and eigenvalue problems. The ScaLAPACK library is built upon a set of communication routines that are themselves a library, called the BLACS (Basic Linear Algebra Communication Subprograms) [6]. Furthermore, with ScaLAPACK comes an additional library called the PBLAS, which can be seen as a parallel distributed version of the BLAS (Basic Linear Algebra Subprograms) [1]. Using the PBLAS simple matrix/vector and more complex operations can be performed in parallel.

Numerical simulations involving massive amounts of calculations are often executed on supercomputers or distributed computing platforms (e.g. clusters and Grids). It is often difficult for non expert users to use such computational infrastructures or even to use advanced libraries over clusters or Grids, because they often require to be familiar with Linux base OS, Grid computing middleware and tools (Glite, Globus, Condor; PBS, OAR; JDL, etc.), parallel programming techniques and packages (MPI; Mpich, Lam MPI; etc.) and linear algebra libraries (ScaLAPACK, LAPACK, BLAS, etc.). In order to overcome the above mentioned problems for linear algebra tasks, a Web portal has been developed, which is continuation of previous work [7].

The purpose of the work is to develop a Grid-aware Web portal for linear algebra tasks with advanced service trading. It provides a seamless bridge between high performance linear algebra libraries such as ScaLAPACK and those users who are not familiar with linear algebra softwares. Note that even when considering a well defined area such as linear algebra, determining the combination of functions (services) of BLAS, LAPACK and ScaLAPACK libraries satisfying a user request is very difficult to determine because many different services and services combinations in the various libraries can fulfill the same requirements. An advanced Grid service trading to compute and find the best service or combination of services that answer the user requirements [8] taking into account the amount of calculations of given expression is used. As a benefit, users do not need to make explicit call to specific services over the Grid (GridRPC, Corba, etc.) and to know the exact name of the service they are looking for: they just describe the expression they want to compute in a given applicative domain.

The experiments of the interface have been carried out on the base of Armenian Grid [9] (SEE Grid site, http:www.cluster.am) and the French National Grid'5000 infrastructures (see http://grid5000.org).

2 Advanced Trading

Identifying the computational service or the combination of computational services satisfying a user request may be quite challenging when a large amount of software is available either on a large scale distributed computing infrastructure or even on a single computer.

A grid service trading approach that computes the service or the combination of services answering an user request [10-11] is used. This trader is based on a semantic description of the available services (often in numerical libraries). Services are described in terms of their algebraic and semantic properties which is equivalent to provide a description of algorithms and objects properties in a given application area.

As a great benefit, users are not required to explicitly proceed to calls to numerical libraries or to grid-services, but instead manipulate high level mathematical expressions.

2.1 Computing the Combination of Services Corresponding to a User's Request

To identify all services and combinations of services that answer the user's problem, the description of the user's problem and the description of all the services are compared, taking into account the properties of the domain (here we focus on dense linear algebra). The realization consists in two steps. In the first step, all available services and combination of available services which answer the user request are computed. In the second step the best one, according to the user's criteria is chosen. It is possible to combine both steps for a better effectiveness.

Using equational unification and more especially the work of Gallier and Snyder in [12] with some modifications in the transformations, an algorithm has been derived to solve the trading problem [10-11].

2.2 Semantic-Based Description of Service: Example in Dense Linear Algebra

The semantic description is similar to algebraic data type description [13], the required information is:

- the types (or sorts);
- the main operators of the specific domain and their signatures (we allow overloading); the operators properties (such as commutativity and associativity) and the equations that link operators.

When considering dense linear algebra and basic operations provided by the BLAS and LAPACK, we define:

- Types: Int, Real, Char, Matrix , . . .
- Operators and their signatures:
  - Addition of matrices: + : *Matrix x Matrix -> Matrix*
  - Multiplication : * : *Real x Matrix -> Matrix; * : Matrix x Matrix -> Matrix*
  - Transpose of a matrix: T : Matrix -> Matrix
  - Identity: I : -> Matrix; Null matrix: O : -> Matrix
  - . . .
- Properties:
  - Commutativity, associativity (can be expressed directly by equations)
  - Neutral / Zero element: a : Matrix I * a = a; a : Matrix O * a = O
  - Distributivity * /+: a : Matrix b : Matrix c : Matrix a * (b + c) = (a * b) + (a * c)

This approach is generic and can be extended to a wide range of domains as soon as a description based on algebraic data type can be defined (experiments have also been carried on in nonlinear optimization and signal processing).

2.3 Description of BLAS and LAPACK Libraries

Describing the Level 3 BLAS or LAPACK procedures is then straightforward using a formalism very similar to the official BLAS specification [1], thanks to a description based on the operators of the domain (term over the algebra defined by this operators). This description has been extended to take into account numerical properties of the matrices and some "non-functional" parameters (like size of the matrices).

In case of matrix-matrix multiplication, symmetry of one of the matrices involved in the operation may lead to select SSYMM rather than SGEMM and similarly in case of triangular matrix that is supported by STRMM. To take into account these properties, subtypes have been introduced in the description:

- Types:
  - Invertible matrices: InvMatrix < Matrix
  - Symmetric matrices: SymetricMatrix < Matrix
  - Triangular matrices: TriangularMatrix < Matrix
  - . . .

We proceed to the same extension on operators and their signatures e.g.:

Multiplication of a symmetric matrix by a scalar: * : *Real x SymetricMatrix -> SymetricMatrix* (the symmetric property is conserved).

SGEMM performs one of the matrix-matrix operations: *C = alpha * op(A) op(B)+ beta * C* where *alpha* and *beta* are scalars, op(A) and op(B) are rectangular matrices of dimensions m-by-k and k-by-n, respectively, C is a m-by-n matrix, and op(A) is A or A^T. In the trader, SGEMM is described by an XML document whose meaning is:

SGEMM(TRANSA:Char, TRANSB:Char, M:Int, N:Int, K:Int, ALPHA:Real, A:Matrix, LDA:Int, B:Matrix, LDB:Int, BETA:Real, C:Matrix, LDC:Int)

*C <- ALPHA * op(TRANSA,A) * op(TRANSB,B) + BETA * C*

Some other information are also store in the XML file to give the meaning of the other parameters (M, N, K, LDA, LDB and LDC).

Among the equations of the domain, we have: op('n', a) = a and op('t', a) = a^T.

2.4 Example 1: Matrix-Matrix Multiplication

Assume that a user wants to solve the following algebraic expression C = A * (B * C), where A, B and C are general matrices.

One combination of services computed by the trader is:

```
p1=O; p2=O;
sgemm('n', 'n', nbRow(p1), nbCol(p1), nbCol(B), 1.0, B, nbRow(B), C, nbRow(C), 1.0, p1, nbRow(p1) ); //p1<-B*C
sgemm('n', 'n', nbRow(p2), nbCol(p2), nbCol(A), 1.0, A, nbRow(A), p1, nbRow(p1), 1.0, p2, nbRow(p2) ); //p2<-A*p1
p2;
```

Where nbRow(X) is the number of rows of the matrix X and nbCol(X) its number of columns.

2.5 Example 2: Matrix Factorization Using LAPACK

We now assume that the available services are the ones from the Level 3 BLAS and some of LAPACK [2]: row interchanges SLASWP, the LU factorization SGETRF. The user wants to solve the linear system with multiple right-hand side members Ax=B (where no property is known about A). One answer computed by the trader is:

```
p1=A;
p2=(ipiv);
sgetrf(nbRow(p1), nbCol(p1), p1, nbRow(p1), p2, (info) );
  //p2<-LU factorization (A= P*L*U)
p3=B;
slaswp(nbCol(p3), p3, nbRow(p3), (k1), (k2), p2, 1 );
  //p3<- line swap B
p4=p3;
strsm('l', 'l', 'n', 'n', nbRow(p4), nbCol(p4), 1.0, p1, nbRow(p1), p4, nbRow(p4) );
  //solve L*x=p3; p4<-x;
p5=p4;
strsm('l', 'u', 'n', 'n', nbRow(p5), nbCol(p5), 1.0,  p1, nbRow(p1), p5, nbRow(p5) );
  //solve U*x=p4; p5<-x;
p5;
```

This is nothing else than the task performed by the LAPACK procedure SGETRS that aims at solving a linear system, which demonstrates the viability of our approach. If the matrix A is known to be symmetric positive definite, a LU factorization will also proposed.

3 Introduction to the Grid-Aware Web Portal

The interface consists of the following three modules:

- A front end module which interacts with users
- The Grid service trading module that computes the service or the combination of services answering a user request. The execution of a service over the grid is nothing else then executing some BLAS, LAPACK or ScaLAPACK procedure or basic manipulation of numerical objects.
- The Solver module which performs computations and uses the DIET middleware for managing executions over the Grid.

3.1 Front End Module

The frontend module is the main module which interacts with users. Linear algebra encompasses methods and concepts that solve many different types of practical problems on the base of numerical objects (number, vector, and matrix). The module allows users to create numerical objects, as well as define and submit various linear algebra expressions. Both uploading and generation mechanisms have been developed for creation of numerical objects by specifying their types: scalars, vectors (ones, zeros, integer, etc.), matrices (triangular, identity, symmetric, etc.). The interface supports different operations (download, delete or change parameters) for already uploaded numerical objects. The names and the paths of the user-defined numerical

objects are saved in the database. Only real and scalar types for numerical objects are implemented in the current experimental version.

3.2 Grid Service Trading Module

The purpose of the Grid trading module is to convert mathematical expressions into a list of procedures calls that answer the user request. In this experimental version, we only consider calling ScaLAPACK procedures but other libraries can easily be added. Executions compute a service or a workflow of services equivalent to the initial mathematical expression. Today the use of LEX [14] and YACC [15] or FLEX and BISON [16] are widely spread among compiler developers. From specifications of the lexical and syntactical structure of a language, using regular expressions and LALR(1) grammars, these tools generate the corresponding analyses. A PHP program based on Bison/YACC has been developed, which checks and compares the parameters of the numerical objects from database and do syntactic and semantic analysis before converting the equation into a sequence of procedure calls.

Our service trading approach is based on a semantic description of services. It allows to compute the service or the combination of services that satisfy a user request (cf paragraph 2.). The input stream of the Grid service trading is a XML description that fully defines the given expression including the parameters and properties. The current module takes the expression from the front-end module, does syntactic and lexical analysis, converts the expression into the corresponding XML format, calls the Grid service trading service and as a result receives the list of LAPACK/ScaLAPACK or BLAS functions that can solve the problem (note that the result may depend on the size of the object manipulated within the expression, e.g. calls to serial or parallel computational services). Since that this approach is based on a semantic description of the libraries, it is easy to increase the number of libraries involved in our environment.

3.3 The Solver Module

Several approaches exist for porting applications to grid platforms e.g. message passing, batch processing, web portals, and Grid-RPC systems. In the case of the Grid-RPC model, clients submit computational requests to a scheduler or an agent that locates one or more servers available over the grid. The DIET Project aims at the development of a scalable middleware with initial efforts focused on distributing the scheduling problem across multiple agents. DIET consists into a set of elements that can be used together to build applications using the Grid-RPC paradigm. This middleware is able to find an appropriate server according to the information given in the client's request. The DIET environment consists of five different components: clients that submit problems to servers, servers that solve problems sent by clients, a database that contains information about software and hardware resources, a scheduler that chooses an appropriate server depending both of the problem sent and the information contained in the database, and finally monitors that get information about the status of the computational resources. The information stored on a server is the list of data it owns (eventually with their distribution and the way to access them), the list of problems that he can solve, and every information concerning its load (CPU capacity, available memory, etc.).

FORTRAN wrappers have been written for the ScaLAPACK functions in order to initialize the process grid, distribute the linear algebra objects on the process grid for the given function, call correspondent ScaLAPACK function and release the process grid, as well as shell scripts have been written for BLAS functions. After getting the list of functions to be called from the previous module, if the size of the data involved in the current call is not so large and can fits into the memory of the current machine, the expression is solved locally, otherwise and in most of cases, the Solver module, which acts as a DIET client, calls the corresponding ScaLAPACK subprogram through a FORTRAN wrapper. The output of the current DIET call can be used as an input of the next procedure call. On Figure 1, we give the detailed structure of the interface.

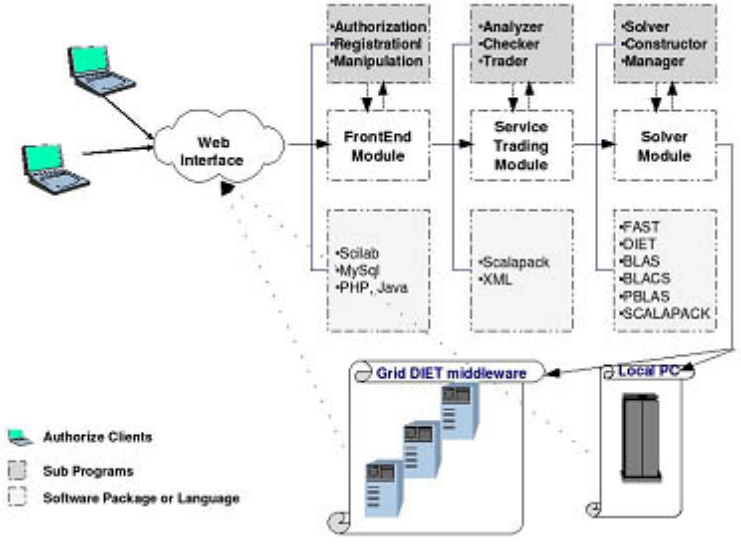


Fig. 1. The Interface structure

4 Examples

To illustrated the possibilities of the framework, an example will be given. As an illustration of the Interface, we return back to the matrix-matrix multiplication example: C = A * (B * C), where A, B and C are general type 10000-by-10000 matrices.

As mentioned above, the interface consists of the Front end, the Grid Service trading and the Solver modules. The interface then processes the request in the following way:

- After successful identification, the user uploads the numerical objects (A, B, C matrices) into the database through the uploading subprogram that includes a numerical object correctness checker. Through the "View numerical objects" subprogram, user can see and edit its own uploaded numerical objects. Only the names and the paths are saved in the database.

- The user enters the mathematical expression to be evaluated that involves the numerical objects initialized as: A * (B * C) and then submits it for execution.
- The syntactic and semantic analyzer subprogram based on Bison/YACC checks and compares the parameters (dimensions and names) of A, B, C matrices from the database and does syntactic and semantic analyzing before converting the equation. If the expression is correct, the analyzer creates an XML file that describes the expression:

```
<requete>
<domainName>LinearAlgebra</domainName>
<term>
 <expression>
  <operatorName>*</operatorName>
  <terms>
   <term>
    <variable>
     <name>A</name>
     <sorte>Matrix</sorte>
    </variable>
   </term>
   <term>
    <expression>
     <operatorName>*</operatorName>
     <terms>
        <term>
         <variable>
          <name>B</name>
          <sorte>Matrix</sorte>
         </variable>
        </term>
        <term>
         <variable>
          <name>C</name>
          <sorte>Matrix</sorte>
         </variable>
        </term>
       </terms>
    </expression>
   </term>
  </terms>
 </expression>
</term>
</requete>
```

- The service trading subprogram takes the XML file (fully defined expression) and converts it into the list of functions that can solve the requested expression. In our case the output is of the service trader is:

PsGEMM ('N', 'N', 10000, 10000, 10000, 1., B, 1, 1, DESCB, C, 1, 1, DESCC, 0., p2, 1, 1, DESCp2) \\ p2= B * C
PsGEMM ('N', 'N', 10000, 10000, 10000, 1 A, 1 1 DESCA, p2, 1, 1, DESCp2, 0., p1, 1, 1, DESCp1) \\ p1= A * p2
p1

Here the PsGEMM procedure, for the PBLAS, performs one of the matrix-matrix operations: C= α*op(A)*op(B)+β*op(C), where α and β are scalars, op(A) and op(B) are rectangular single precision real matrices of dimensions m-by-k and k-by-n respectively, C is a m-by-n matrix, and op(A) is a A or A^T.

- A list of FORTRAN wrapers have been developed for all PBLAS and BLAS functions. The FORTRAN programs initialize the process grid, distribute the matrix on the process grid, call correspondent PBLAS routine and then release the process grid. Input parameters of FORTRAN subprograms include the paths and files names of numerical objects. The output is stored saved into a standard temporary ASCII file that can be used as an input for the next FORTRAN subprogram. Taking into account the dimensions of the numerical object for given routine, the solver module defines where it is wise to execute the ScaLAPACK subprogram and then executes it. If the dimensions of the current call are not too large and can fit into the memory of local PC, the expression is solved locally by calling correspondent BLAS function, otherwise and most of cases the Solver module performs executions over the grid using DIET and ScaLAPACK.
- After successful executing all ScaLAPACK subprograms, the last output is saved into the database. The final result is accessible from the interface and user can download it if necessary.

5 Conclusion

The full implementation of our environment for scientific computing over the Grid will allow users, who are not familiar with the parallel programming technologies and software tools (Grid & Cluster middlewares, MPI, JAVA, Unix OS, Open PBS, Condor, ScaLAPACK, etc.), to create and submit their programs over a Web interface that hides all details of the underlying distributed infrastructure. Such an approach can be extended to any computational Grid that supports DIET (or another GridRPC type of middleware) and linear algebra libraries such as ScaLAPACK. It can be easily extended to any application area where software libraries are available. Note that it is also a nice environment for demonstrating how computations can be transparently – from the user point of view – performed over the Grid.

Acknowledgment

The joint work has been done within the framework of INTAS YS Post Doctoral Fellowship (05-109-4390). It has also be supported the French LEGO (ANR-05-CIGC-11) and ISTC A-1451 Projects. We are very grateful to the GRID'5000 initiative for providing access to the nation-wide experimental grid platform.

References

1. Blackford, S., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., Heroux, M., Kaufman, L., Lumsdaine, A., Petitet, A., Pozo, R., Remington, K.: An Updated Set of Basic Linear Algebra Subprograms. ACM Trans. Math. Soft., pp. 135-151, 28-2 (2002)
2. Anderson, E., Bai, Z., Bischof, C., Blackford, L.S., Demmel, J., Dongarra, J.J., Du Croz, J., Hammarling, S., Greenbaum, A., McKenney, A., Sorensen, D.: LAPACK Users' guide. In: Society for Industrial and Applied Mathematics, 3rd edn., Philadelphia, PA, USA (1999)
3. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK: A Linear Algebra Library for Message-Passing Computers. In: SIAM Conference on Parallel Processing (March 1997)
4. Bunks, J.P., Chancelier, F., Delebecque, C., Gomez, M., Goursat, R., Nikoukhah, S.: Steer, Engineering and Scientific Computing with Scilab, Hardcover, p. 491 (1999) ISBN: 978-0-8176-4009-5
5. Caron, E., Desprez, F.: DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. International Journal of High Performance Computing Applications 20(3), 335–352 (2006)
6. Dongarra, J., Whaley, R.: LAPACK Working Note 94: A User's Guide to the BLACS v1.0, University of Tennessee Computer Science Technical Report, UT-CS-95-281 (Version 1.1) (March 1995) (updated May 5, 1997)
7. Astsatryan, H., Daydé, M., Hurault, A., Pantel, M., Caron, E.: On defining a Web Interface for Linear Algebra Tasks over Computational Grids. In: International Conference on Computer Science and Information Technologies (CSIT 2007), Yerevan, Armenia (2007) 24/09/2007-28/09/2007
8. Daydé, M., Hurault, A., Pantel, M.: Semantic-based Service trading: Application to Linear Algebra. In: Daydé, M., Palma, J.M.L.M., Coutinho, Á.L.G.A., Pacitti, E., Lopes, J.C. (eds.) VECPAR 2006. LNCS, vol. 4395, pp. 622–633. Springer, Heidelberg (2007)
9. Astsatryan, H.V., Shoukourian, Y., Sahakyan, V.: Creation of High-Performance Computation Cluster and DataBases in Armenia. In: Proceedings of the Second International Conference on Parallel Computations and Control Problems (PACO 2004), Moscow, Russia, October 4-6, 2004, pp. 466–471 (2004) ISBN: 5-201-14974-X
10. Hurault, A., Daydé, M., Pantel, M.: Advanced service trading for scientific computing over the grid. The Journal of Supercomputing (to appear, 2008)
11. Hurault, A.: Courtage sémantique de services de calcul, PhD Thesis, INPT, Toulouse (December 2006)
12. Gallier, J.H., Snyder, W.: Complete Sets of Transformations for General E-Unication. Theor. Comput. Sci. 67(2-3), 203–260 (1989)
13. Guttag, J.V., Horning, J.J.: The algebraic specication of abstract data types. Acta Inf. 52, 27–52 (1978)
14. Lesk, M.E.: Lex - a lexical analyzer generator, Computing Science Technical Report 39, AT&T Bell Laboratories, Murray Hill, NJ (1975)
15. Johnson, S.C.: Yacc: yet another compiler compiler. CS Technical Report #32, Bell Telephone Laboratories, Murray Hill, NJ (1975)
16. Aaron Montgomery, Using Flex and Bison, `http://www.mactech.com/articles/mactech/Vol.16/16.07/UsingFlexandBison/`

Resource Matching in Non-dedicated Multicluster Environments*

J. Ll. Lérida[1], F. Solsona[1], F. Giné[1], J.R. García[2], and P. Hernández[2]

[1] Departamento de Informática e Ingeniería Industrial, Universitat de Lleida, Spain
{jlerida,francesc,sisco}@diei.udl.cat
[2] Departamento de Arquitectura y Sistemas Operativos, Universitat Autònoma de Barcelona, Spain
jrgarcia@aomail.uab.es, porfidio.hernandez@uab.cat

Abstract. We are interested in making use of Multiclusters to execute parallel applications. The present work is developed within the M-CISNE project. M-CISNE is a non-dedicated and heterogeneous Multicluster environment which includes MetaLoRaS, a two-level MetaScheduler that manages the appropriate job allocation to available resources.

In this paper, we present a new resource-matching model for MetaLoRaS, which is aimed at mitigating the degraded turnaround time of co-allocated jobs, caused by the contention on shared inter-cluster links. The model is linear programming based and considers the availability of computational resources and the contention of shared inter and intra-cluster links. Its goal is to minimize the average turnaround time of the parallel applications without disturbing the local applications excessively and maximize the prediction accuracy.

We also present a parallel job model that takes both computation and communication characterizations into account. By doing this, greater accuracy is obtained than in other models only focused on one of these characteristics.

Our preliminary performance results indicate that the linear programming model for on-line resource matching is efficient in speed and accuracy and can be successfully applied to co-allocate jobs across different clusters.

1 Introduction

A Multicluster system has a network topology made up of interconnected clusters, limited to a campus- or organization-wide network. There are collections of several clusters formed by commodity workstations in many laboratories, Universities, and research centers. The main goal of the present work is to make use of wasted computational resources of non-dedicated and heterogeneous Multiclusters to execute parallel applications efficiently without disturbing the local applications excessively.

In order to manage the collective computational power of a Multicluster efficiently, special scheduling mechanisms are required to select and map jobs to available resources. We refer to these schedulers as MetaSchedulers. In general, we consider a MetaScheduler to be the software that decides where, when, and how to schedule jobs in

* This work was supported by the MEyC-Spain under contract TIN2007-64974.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 160–173, 2008.

a Multicluster. In previous works [11,12], we presented MetaLoRaS, an efficient MetaScheduler made up of a queuing system with two-level hierarchical architecture for a non-dedicated Multicluster. The most important contribution was the effective cluster selection mechanism, based on the estimation of the job turnaround time. Parallel applications were assigned to the clusters where the minimum turnaround time was obtained. MetaLoRaS was globally aware of the state of the Multicluster and worked in conjunction with each individual cluster's local schedulers.

A Multicluster is distinguished from a traditional computational grid in that the Multicluster utilizes a dedicated interconnection network between cluster resources with a known topology and predictable performance characteristics. This kind of networking infrastructure allows for the possibility of mapping jobs across cluster boundaries in a process known as co-allocation or multisite scheduling. Co-allocation of parallel jobs is considered in this paper, as is minimizing their execution time, this being the desired goal.

Previous work in the area of job co-allocation has tended to characterize jobs based only on communication or computation models. Ernemann and Jones [6,10] describe how schedulers designed to allocate node resources across cluster boundaries can result in rather poor overall performance over a wide range of workload characterizations and Multicluster configurations when co-allocated jobs contend for inter-cluster network bandwidth. In order to overcome these situations, our model is based on co-allocating job tasks to avoid both the communication saturation of inter-cluster links and the overloading of Multicluster nodes, which is not considered in these works. In [5,9,4,10] only communication models are presented, being useful to evaluate the system performance instead of taking online scheduling decisions and accurate predictions about the execution time.

The essence of our MetaScheduling model is to solve the resource matching as an integer-programming problem. Previous work [3,13] illustrates the benefits of using integer programming techniques to solve scheduling problems. However, Naik [13] provides a globally optimal for the system performance assuming that workload is known, and Banino [3] centered on time-sharing scheduling solutions difficult to implement in practice.

The present work aims to extend the works presented in [10,14] and [11,8], creating a MetaScheduling model which takes into account the effect of co-allocation on both computing and communication times. By doing so, we are able to mitigate the negative effect on co-allocated jobs, improving the prediction accuracy of the turnaround time estimation of parallel jobs. This in turn increase the system performance by improving the prediction-like scheduling system. Furthermore, the model takes into account the resource occupancy and capacity of the forming non-dedicated Multicluster nodes. This fact guarantees low impact on the performance of local user applications.

The rest of the paper is organized as follows. In section 2, we present the characterization of both, the Multicluster environment and parallel jobs. In section 3, the integer programming model for matching parallel applications in Multicluster systems is presented. The applicability of the model and the goodness when applied in a real Multicluster system is evaluated in Section 4. Finally, the conclusions and future work are detailed.

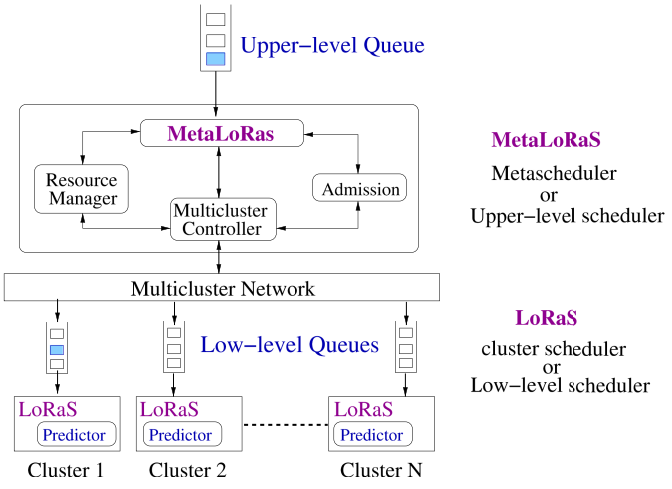


Fig. 1. Multicluster Architecture

2 Multicluster Environment

In [12] we proposed a Multicluster platform. The jobs arriving in the Multicluster enter the Upper-level Queue awaiting scheduling by the MetaScheduler, named *MetaLoRaS*. *MetaLoRaS* assigns jobs to the cluster with the minimum estimate of turnaround time. The estimation is obtained by each local cluster or Low-level scheduler, named *LoRaS* (Long Range Scheduler). *LoRaS* [7] is a space-sharing scheduler with an efficient turnaround predictor [8].

MetaLoRaS is made up of five components (see Fig. 1). These are the *Upper-level Queue* (a queuing system), the Multicluster scheduler (named *MetaLoRaS*), the *Admission system*, the *Resource Manager* and the *Multicluster Controller*.

MetaLoRaS is the Multicluster scheduling system. It is responsible for selecting the next job to be executed from the *Input Queue* (the entry point of parallel jobs), and also the cluster where this job will be executed. The part of *MetaLoRaS* responsible for assigning jobs to clusters is denoted as *Resource Matcher* (RM).

The *Admission System* is responsible for admitting new jobs into the system. This module will accept the new job whenever its required resources are satisfied. If not, the job is discarded. The specified resources are the number of workstations, the Memory size and the per-node bandwidth. It is possible to specify different resource limits in each cluster.

The *Multicluster Controller* collects real time information about the state of each cluster. If an event occurs in one cluster (job start, finish), the *Multicluster Controller* is notified of such a change. The *LoRaS* system is responsible for notifying the *Multicluster Controller* about the cluster state changes.

The *Resource Matcher* (RM) has been designed as an Integer programming approach. The RM is responsible for obtaining a snapshot of the state of the resources

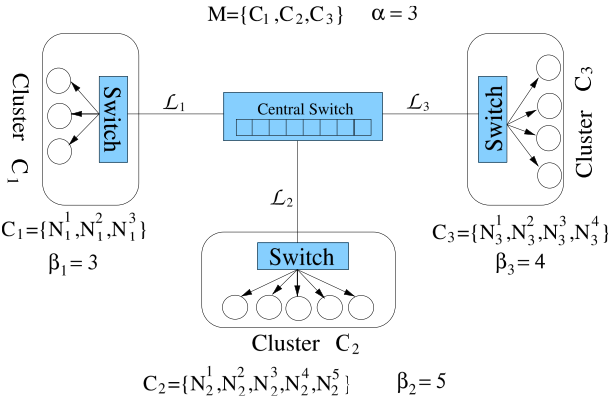


Fig. 2. Multicluster topology

from the *Multicluster Controller* and for generating a mapping solution that will be used by the MetaScheduler *(MetaLoRaS)*. To do this, the RM performs the following functions: (1) it accepts a job matching request through the MetaScheduler, (2) requests the current status of the Multicluster from the *Multicluster Controller*, (3) obtains the parallel application information, (4) submits the parallel application and the Multicluster status information to a mixed integer programming solver and (5) maps the job accordingly to the results obtained in step 4.

Job co-allocation consists of mapping jobs across cluster boundaries. Co-allocation is necessary when a job requires more nodes than the ones available on each particular cluster, but collectively there may be enough available nodes elsewhere in the Multicluster to accommodate such a job. There are situations where despite having enough available nodes in a particular cluster, it may be better to take advantage of remote resources, because they are more powerful or they are the more appropriate for the nature of the parallel job. The *Resource Matcher* (RM) is responsible for deciding if the job will be co-allocated across multiple clusters or mapped exclusively onto one cluster. Scheduling decisions are based on minimizing the job execution time, despite the jobs are exclusively assigned to an unique cluster or across multiple clusters.

2.1 Problem Statement

We are interested on Multiclusters defined as a collection of arbitrary sized clusters with heterogeneous resources. Each cluster has its own internal switch. Clusters are connected to each other by single dedicated links by means of a central switch.

Formally, a Multicluster $M=\{C_1..C_\alpha\}$ can be defined as a system comprised of α heterogeneous clusters interconnected by means of dedicated links (see Fig. 2). Each Cluster C_i ($1 \leq i \leq \alpha$) is also made up by β_i nodes, this is $C_i=\{N_i^1..N_i^{\beta_i}\}$. $\mathscr{L}$ is the set of inter-cluster links ($\mathscr{L}=\{\mathscr{L}_1..\mathscr{L}_\alpha\}$), and $L=\{L_i\}=\{L_i^k,\ 1\leq i \leq\alpha$ and $1\leq k \leq\beta_i\}$, is the set of intra-cluster links, where L_i^k denote the intra-cluster link between node k and the switch of Cluster C_i. We suppose that network bandwidth and latency of inter-cluster links are better than the intra-cluster ones.

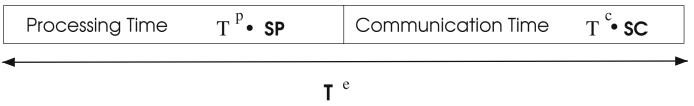


Fig. 3. Execution Slowdown

The model assumes that the jobs follow a BSP (Bulk Synchronous Parallel) model. A BSP job is comprised of coarse or medium grained tasks that require a fixed number of processors (one per task) during their lifetime. The size of their component tasks is generally similar. In addition, each task is comprised of various iterations in which computation alternates with communication and synchronization phases. The job assignment is static, that is, once the job is mapped into a particular set of nodes, no more re-allocations are performed. Additionally, jobs can be co-allocated in a Multicluster by allocating nodes from different clusters to the same job in order to better meet the collective needs across the Multicluster.

We define the job's execution time, T^e (see Fig. 3), as follows:

$$T^e = T^p \cdot SP + T^c \cdot SC, \tag{1}$$

where T^p and T^c are the processing and communicating times in a dedicated environment. In a real situation, due to the heterogeneity and the non-dedicated property of the resources, T^p and T^c may be lengthened by SP and SC, the processing and communication slowdown respectively.

2.2 Processing Characterization

In a heterogeneous and non-dedicated environment, the computing power and its availability can provoke different processing capabilities of the constituent nodes. The current work presents solutions for measuring these factors and studying their effect on the execution time of the co-allocated jobs.

In a heterogeneous environment, we must take into account the computing power differences between the processor units that form the Multicluster. According to [5], we define the relative Power weight (P_i^k) of the cluster i node k ($1 \leq i \leq \alpha$ and $1 \leq k \leq \beta_i$), as the computing power ratio of such node with respect to the most powerful node of the Multicluster. The P_i^k range is $0 < P_i^k \leq 1$. $P_i^k = 1$ means that cluster i node k is the most powerful node in the Multicluster. We obtain the relative computing power of each node by averaging various relative power measurements with different applications.

Local and even parallel jobs executing on the cluster lower the performance of new parallel jobs.The model takes this situation into account by sampling the availability of the computing resources. As was shown in [14], we can obtain an effective measurement of the CPU availability by relating the average of the number of process in the system and the CPU occupancy. We define the Availability of the cluster i node k (A_i^k) as the percentage of CPU occupancy. $A_i^k \simeq 0$ when 100% of the CPU is occupied and $0 < A_i^k \leq 1$ otherwise.

We define the Effective Power weight of cluster i node k (Γ_i^k) as the product between the relative Power weight and the Availability of such a node. Formally, Γ_i^k is defined as follows:

$$\Gamma_i^k = P_i^k \cdot A_i^k, \tag{2}$$

where $\Gamma_i^k = 1$ means that cluster i node k has the full capacity to run the tasks at full speed. When $0 < \Gamma_i^k < 1$, the node k of cluster i is unable to execute the task at full speed. Therefore, the processing slowdown of such a node (SP_i^k) is inversely proportional to its Effective Power weight, $SP_i^k = (\Gamma_i^k)^{-1}$.

As in our model we assume that each job task is generally similar in size and they are executing separately, the job execution time is defined as the elapsed execution time of the slowly task. Thus, the processing slowdown can be obtained by taking the node with the lowest Effective Power weight into account, or in other words, the node with the maximum slowdown. According to this, we formally define the slowdown of processing time (SP) in function of the slowdown obtained by each allocated node as follows:

$$SP = max\{SP_i^k, 1 \leq i \leq \alpha \, and \, 1 \leq k \leq \beta_i\} \tag{3}$$

2.3 Communication Characterization

Communication characterization is based on the model described by Jones in [10] for homogeneous and dedicated environments. We provide resource heterogeneity to Jone's model. Furthermore, we add the ability to take into account the effect of the local workload on the co-allocated applications.

We assume that the parallel jobs follow an all-to-all communication pattern periodically throughout their execution, one of the most frequently used in parallel processing. Each task of a given job j is characterized by an average per-node bandwidth metric, $PNBW^j$, consisting of the communication needs for job j.

In co-allocation cases, nodes can communicate across cluster boundaries. This communication will require a certain amount of bandwidth on the inter-cluster network links. Saturation of inter-cluster links reduces job performance drastically. In order to determine when the inter-cluster links become saturated, we must identify how much bandwidth a job will require, and more precisely, each forming task job.

We define BW_i^j (equation 4) as the amount of bandwidth required by job j on inter-cluster link i ($1 \leq i \leq \alpha$). Formally:

$$BW_i^j = \left(n_i^j \cdot PNBW^j\right) \cdot \left(\frac{n_T^j - n_i^j}{n_T^j - 1}\right), \tag{4}$$

where n_T^j is the total number of nodes required by job j, and n_i^j is the number of nodes allocated to job j on the cluster C_i. The first factor of the equation is the total bandwidth required by all the nodes associated with job j on cluster C_i. The second factor represents the communication percentage of job j with other cluster nodes (not in C_i), that will use the inter-cluster link i.

Each communication link i is characterized by a maximum bandwidth rating, BW_i^{max}. We define the saturation degree of an inter-cluster link i (BW_i^{sat}) as the ratio between

the maximum bandwidth and the total bandwidth required by the jobs that share the link i. Formally:

$$BW_i^{sat} = \frac{BW_i^{max}}{BW_i^{consumed} + BW_i^j}, \tag{5}$$

where $BW_i^{consumed}$ is the bandwidth occupied by other local or parallel applications in the link i. When $BW_i^{sat} \geq 1$, the link i is *not saturated*. Otherwise, when $0 \leq BW_i^{sat} < 1$ the link i is *saturated*.

A job j using a saturated inter-cluster link i will experience a communicating slowdown inversely proportional to the saturation degree of such a link i. Formally:

$$SC_i = \left(BW_i^{sat}\right)^{-1} \tag{6}$$

If any, the most saturated inter-cluster link will determine the communication slowdown of the co-allocated job. We define the communication slowdown of a job j (SC) as the maximum communication slowdown of such job in each allocated inter-cluster link. Formally:

$$SC = max_i\{SC_i, 1 \leq i \leq \alpha\} \tag{7}$$

3 IP Matching Model

Integer Programming (IP) is a technique for solving certain kinds of problems: maximizing or minimizing the value of an objective function subject to some constraints. The objective function and constraints are linear expressions. In the following, we describe our resource-matching approach based on mixed-integer programming techniques.

The problem to be solved in the IP model is the matching of jobs in a Multicluster environment, while avoiding the negative effects of sharing the communication links and processing resources. To do this, the IP model must represent the *job matching request* (specifying their resource requirements) and the state of the Multicluster resources (*Multicluster State*) in order to search for an optimal solution.

The *job matching request* specifies the job requirements as the number of tasks, amount of Memory, per-node bandwidth and the ratio between computation and total execution time. Multicluster nodes without enough Memory are discarded.

The *Multicluster State* comprises the following information of every node: CPU and Memory availability, and both maximum capacity and availability of the intra-cluster communication links. The corresponding inter-cluster information is obtained from the intra-cluster one and the previous job assignments. Only periodic samples of the Multicluster nodes is necessary.

The *Resource Matcher* maps the jobs by minimizing the job execution time. Jobs can be mapped across cluster boundaries. The obtaining of this minimum is performed by means of the Integer Programming solver of CPLEX [1], by using the "Branch and Bound" algorithm. Obviously, this is a well known NP-complete problem. The interest of this work is centered in the definition of heuristics and constraints which delay the exponential time-cost with the number of Multicluster nodes as much as possible.

Input arguments:

1. j: job to be matched.
2. τ^j: number of tasks making up job j.
3. $PNBW^j$: per-node bandwidth requirement for the job j.
4. $M = C..C_\alpha$: Multicluster composition.
5. $\mathscr{L}$ and L=$\{L_i\}$=$\{L_i^k, 1 \leq i \leq \alpha$ and $1 \leq k \leq \beta_i\}$: set of inter- and intra- cluster links.
6. Γ_i^k: Effective Power weight for the cluster i node k ($1 \leq i \leq \alpha$ and $1 \leq k \leq \beta_i$).
7. BW_i^{av}: available bandwidth for each inter-cluster link $\mathscr{L}_i$, $1 \leq i \leq \alpha$.
8. BW_i^{max}: maximum bandwidth for each inter-cluster link $\mathscr{L}_i$, $1 \leq i \leq \alpha$.

Output parameters:

9. X_i^k, $1 \leq i \leq \alpha$ and $1 \leq k \leq \beta_i$: boolean variable associated to cluster i node k. X_i^k=1 if job j is matched to cluster i node k, and 0 otherwise.
10. SP: processing slowdown. $SP = max\{SP_i^k, 1 \leq i \leq \alpha \, and \, 1 \leq k \leq \beta_i\}$.
11. SC: inter-cluster link communication slowdown. $SC = max_i\{SC_i, 1 \leq i \leq \alpha\}$.

Objective Function:

12. $min\{T^e\}$

Constraints:

13. Gang matching.
14. Non inter-cluster link saturation.

Fig. 4. Model Definition

3.1 Model Definition

An integer-programming model includes input parameters, variables, a set of constraints on the value of the variables, and an objective function. The goal of the model is to find values for every variable so that all constraints are satisfied and the value of the objective function is maximized or minimized.

The input parameters, objective function and constraints of the model presented in this work, are shown in figure 4.

Given a job j, this model finds the best feasible match between the job and the resources taking the heterogeneity and the availability of the resources into account along with the requirements of the job j.

The model accepts as input argument a job j, defined by the number of tasks (τ^j) and the per-node bandwidth ($PNBW^j$). Another group of input arguments are the ones characterizing the Multicluster (M). The variable Γ_i^k defines the Effective Power weight of each node, and the variables BW_i^{av} and BW_i^{max} are the available and maximum bandwidths respectively of the inter-cluster links ($\mathscr{L}$).

The output parameter X_i^k is a boolean variable informing about the mapping of the job j. Other outputs are SP and SC, defined in sections 2.2 and 2.3 respectively. The constraints and the objective function are defined below.

3.2 Constraints

The IP model comprises two constraints, the Gang matching and the non-saturation of inter-cluster links. As major network performance is supposed to inter-cluster links, their constraints also includes the saturation of the intra-cluster ones. Next the two constraints are studied separately.

Gang Matching This constraint ensures that we allocate all the required resources of the parallel job j. In other words, each task is allocated to one processor. The gang matching constraint is formalized with the linear equation 8.

$$\sum_{1 \leq i \leq \alpha, 1 \leq k \leq \beta_i} X_i^k = \tau^j, \tag{8}$$

where τ^j is the number of tasks making up job j and X_i^k is equal to 1 if a task in job j is assigned to cluster i node k. This constraint guarantees the assignment of every task making up job j.

Non Inter-Cluster Link Saturation Non-saturation of the inter-cluster links ensures that the bandwidth consumed by the mapping does not exceed the total available bandwidth capacity of the inter-cluster links. This constraint avoids the saturation of inter-cluster links. We formalize this constraint with the equation 9.

$$SC \leq 1, \tag{9}$$

where $SC = max_i\{SC_i, 1 \leq i \leq \alpha\}$ is the maximum slowdown of the inter-cluster links used by job j. The inter-cluster link slowdown, SC_i, was calculated by means of equation 6, explained in section 2.3.

3.3 Objective Function

The objective function defines the quality of a solution when multiple feasible solutions exist. The matching solver uses the objective function to select the best matching solution. In the present work, we are interested in obtaining the minimum execution time for parallel jobs (T^e), defined in section 2.1 equation 1. Accordingly, the objective function is formalized by equation 10.

$$min\{T^e\} \tag{10}$$

4 Experimentation

To study the efficiency of the proposed model we made a great range of tests modifying the amount of resources, their utilization, and the parallel applications characterization. Moreover, we tested the prediction accuracy of the execution time executing parallel applications in a real environment.

The real environment was a Multicluster made up of 2 non-dedicated clusters (CLUSTER1 and CLUSTER2). CLUSTER1 was made up of ten 3-GHz uni-processor workstations with 1GB of RAM, interconnected by a 1-Gigabit network. CLUSTER2 was a heterogeneous cluster made up of ten workstations, five 3-GHz uni-processor with 1GB of RAM and 1-Gigabit network link, and five 3-GHz multiprocessor with 512MB of RAM and 100Megabit network link.

To carry out the experimentation, local and parallel applications need to be defined. The local workload was represented by a synthetic benchmark (named *local_bench*) that can emulate the usage of 3 different resources: CPU, Memory and Network traffic. The use of these resources was parametrized in a real way. According to the values obtained by collecting the user activity in an open laboratory over a couple of weeks, *local_bench* was modeled to use 15% CPU, 35% Memory and a 0.5KB/sec LAN, in each node where it was executed.

We selected two parallel applications, which follows the BSP model, from the NAS parallel benchmarks suite [2]: MG (Multigrid) and IS (Integer Sort). However, the two jobs had different communication patterns and processing/communication needs at each iteration. These parallel jobs were characterized by the number of tasks, the computation time, and the size of their communications.

To study the effect of the constraints on the efficiency of our proposal we defined three different models with different constraint specifications:

Optimal. This approach obtains the optimal solution, looking for the minimum effective slowdown of parallel applications. This model allows the utilization of saturated links. It aims to obtain the best mapping by taking the characterization of parallel jobs and the resource availability into account.

Non-Saturated. In this model the non inter-cluster link saturation constraint was applied. The solver attempts to minimize the execution time of the mapping solutions that will not saturate any inter-cluster link.

Non-Saturated with Non-Optimization. This model does not looks for the optimal solution. Thus, the first solution that avoids the inter-cluster links saturation is returned. This model is thought to be useful with Multiclusters with a high number of resources, where the obtaining of the optimal solution is excessively expansive.

To evaluate the efficiency of our mixed integer programming approach we compare the elapsed time of the *Resource Matcher* to obtain a feasible solution (by means of the CPLEX solver [1]) for different types of parallel jobs and local activity requirements, with different amount of computational resources and inter-cluster links. The per-node bandwidth requirements of the parallel task ($PNBW^j$) was varied from 25% to 75%. The number of workstations with local activity was varied from 0 to 75%.

4.1 Performance Results

First or all, we want to compare the effect of different processing and communication loads on the Optimal and Non-Saturated models.

Figure 5 shows the resulting communication slowdown obtained by the matching solver. As can be seen in Figure 5(right), the Non-Saturated model ensures the non inter-cluster links saturation. Otherwise, in the Optimal model, figure 5(left), the slowdown grows quickly with the network requirements of the parallel application ($PNBW^j$). The local activity has less effect in both models.

Figure 6 shows the effects of $PNBW^j$ and the local activity in the obtaining of the resource matching (by the solver). The behaviour of the models are opposed. The Non-Saturated model is more time-costly than the Optimal one by increasing the nodes with

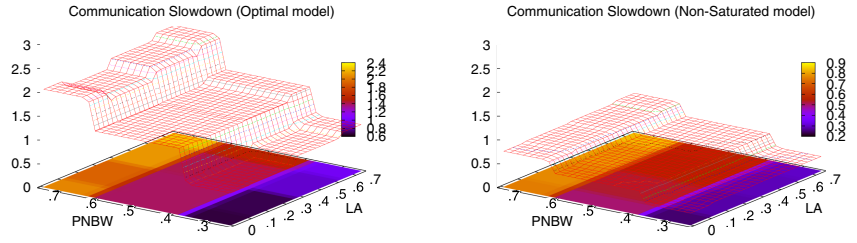


Fig. 5. Communication Slowdown varying $PNBW^j$ and the local activity (LA)

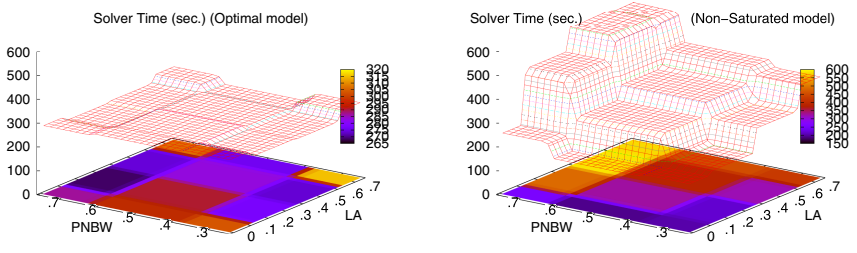


Fig. 6. Solver time varying $PNBW^j$ and the local activity (LA)

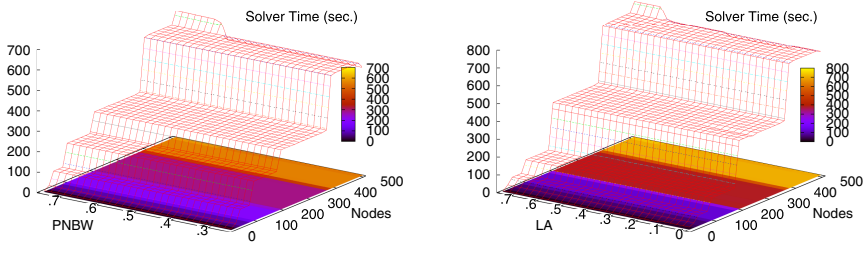


Fig. 7. Solver time. (left) $PNBW^j$ vs. nodes (right) LA vs. nodes.

local activity and the $PNBW^j$. We can observe as in the Optimal model, the bandwidth requirements has a smooth effect on the solver behaviour. Meanwhile, for the Non-saturation model, figure 6(right), the solver response time grows quickly with the bandwidth requirements. This is produced because in the Non-saturated model there are less valid solutions, an the obtaining of one of them is more difficult in time.

Figure 7 shows the solver response time of the Optimal model, by varying the number of nodes jointly with $PNBW^j$ (left) and the local activity (right). It can be appreciated as the predominant parameter in this model is the number of nodes. These results corroborates the ones obtained in Fig. 6(left).

Figure 8 shows the impact of the number of inter-cluster links on the solver response time for different constraints. To study this relationship we fixed the number of nodes per cluster (8 nodes) and ranged between 2 to 64 the number of clusters (inter-cluster

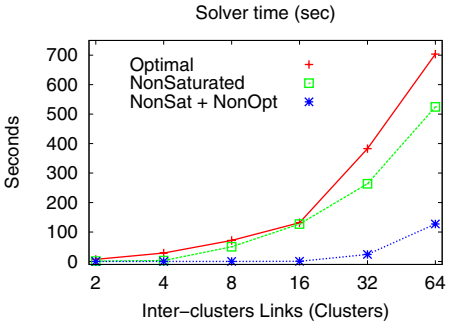


Fig. 8. Resource Matching Time

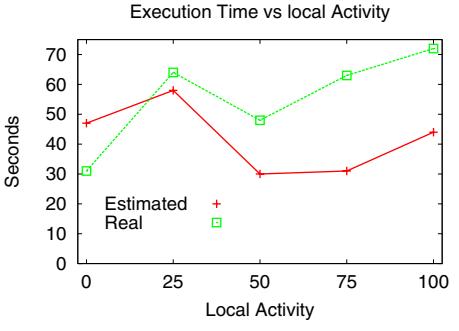


Fig. 9. Estimated vs. Real times

links). The number of constraints in the model have a direct impact on the solver response time. The obtained results indicate that Optimal and Non-Saturated models have a correct behavior for a reasonable number of resources. In our case, below 16 inter-cluster links with 8 nodes per cluster (128 nodes), the response time never overtake one minute. Above this threshold it is advisable the use of the Non-Saturated with Non-Optimization model.

4.2 Prediction Accuracy

In order to evaluate the prediction accuracy of the IP model, we compared the estimations produced by the solver with the real executions of IS and MG. Both benchmarks were executed multiple times with different number of tasks and different local activity situations. Solver times were obtained by using the Optimal model.

The obtained results (see Fig. 9) are very hopeful. Despite the differences between the estimated and real times, we thought that the estimated times can be corrected by applying some sort of correction mechanism, because the two lines have a similar shape. This is the most interesting field to be investigated in the future.

5 Conclusions and Future Work

In the present work we have presented a resource matching mechanism based on integer programing, for non-dedicated and heterogeneous Mulsticluster systems. The model fits efficiently both computation and communication parallel requirements to available Mulsticluster resources by considering the sharing of resources between parallel and local applications.

The results show that, using mixed integer programing, we can model different resource matching situations in a flexible way, and solve them efficiently. As we shows, the number of resources has a great impact on the solver response time. It is important to develop mechanisms to adapt dynamically to inter-arrival job rate, number of resources, etc. The IP model described in the present work allows to adapt the scheduling system to these situations dynamically.

Future work is directed towards the search for a correction factor of the estimates. We also will investigate regression models in the obtaining of the Multicluster State. Due to the intrinsic dynamism of non-dedicated Multiclusters, their state change very quickly, and the on-time monitoring used in this work does not reflect this situation correctly.

In this study, we considered one job at a time. In a further work, we wish to consider the matching problem for multiple jobs, in order to avoid solving large optimization problems achieving a global optimal. Moreover, this matching scheme will allow the matching solver to apply new objective functions based, for example, on throughput or load balancing.

On the other hand, we want to compare the benefits on the system performance obtained with the use of the mixed-integer programing approach, with other meta-scheduling mechanisms based only on partial information about the communications or computation capabilities.

References

1. Cplex (June 30, 2007), `http://www.ilog.com`
2. Bailey, D.H., Dagum, L., Barszcz, E., Simon, H.D.: NAS parallel benchmark results. In: Supercomputing, pp. 386–393 (1992)
3. Banino, C., Beaumont, O., Carter, L., Ferrante, J., Legrand, A., Robert, Y.: Scheduling strategies for master-slave tasking on heterogeneous processor platforms. IEEE Transactions on Parallel and Distributed Systems 15(4), 319–330 (2004)
4. Bucur, A.I., Epema, D.H.: The performance of processor co-allocation in multicluster systems. In: CCGRID 2003: Proceedings of the 3st International Symposium on Cluster Computing and the Grid, Washington, DC, USA, p. 302. IEEE Computer Society, Los Alamitos (2003)
5. Du, X., Zhang, X.: Coordinating parallel processes on networks of workstations. Journal of Parallel and Distributed Computing 46(2), 125–135 (1997)
6. Ernemann, C., Hamscher, V., Streit, A., Yahyapour, R.: Enhanced algorithms for multi-site scheduling (2002)
7. Hanzich, M., Giné, F., Hernández, P., Solsona, F., Luque, E.: Cisne: A new integral approach for scheduling parallel applications on non-dedicated clusters. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 220–230. Springer, Heidelberg (2005)

8. Hanzich, M., Giné, F., Hernández, P., Solsona, F., Luque, E.: Using on-the-fly simulation for estimating the turnaround time on non-dedicated clusters. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 177–187. Springer, Heidelberg (2006)
9. Javadi, B., Abawajy, J.: Performance analysis of heterogeneous multi-cluster systems. In: ICPPW 2005: Proceedings of the 2005 International Conference on Parallel Processing Workshops (ICPPW 2005), Washington, DC, USA, pp. 493–500. IEEE Computer Society, Los Alamitos (2005)
10. Jones, W.M., Ligon, W.B., Pang, L.W., Stanzione, D.: Characterization of bandwidth-aware meta-schedulers for co-allocating jobs across multiple clusters. The Journal of Supercomputing V34(2), 135–163 (2005)
11. Lérida, J.L., Solsona, F., Giné, F., Hanzich, M., García, J.R., Hernández, P.: Metaloras: A re-scheduling and prediction metascheduler for non-dedicated multiclusters. In: Cappello, F., Herault, T., Dongarra, J. (eds.) PVM/MPI 2007. LNCS, vol. 4757, pp. 195–203. Springer, Heidelberg (2007)
12. Lérida, J.L., Solsona, F., Giné, F., Hanzich, M., Hernández, P., Luque, E.: Metaloras: A predictable metascheduler for non-dedicated multiclusters. In: Guo, M., Yang, L.T., Di Martino, B., Zima, H.P., Dongarra, J., Tang, F. (eds.) ISPA 2006. LNCS, vol. 4330, pp. 630–641. Springer, Heidelberg (2006)
13. Naik, V.K., Liu, C., Yang, L., Wagner, J.: Online resource matching for heterogeneous grid environments. Ccgrid 2, 607–614 (2005)
14. Wolski, R., Spring, N., Hayes, J.: Predicting the CPU availability of time-shared unix systems on the computational grid. Cluster Computing 3(4), 293–301 (2000)

A Parallel Incremental Learning Algorithm for Neural Networks with Fault Tolerance

Jacques M. Bahi[1], Sylvain Contassot-Vivier[2], Marc Sauget[1,3], and Aurélien Vasseur[3]

[1] LIFC, University of Franche-Comté, Belfort, France
{jacques.bahi,marc.sauget}@iut-bm.univ-fcomte.fr
http://info.iut-bm.univ-fcomte.fr/
[2] LORIA, University Henri Poincaré, Nancy, France
sylvain.contassotvivier@loria.fr
http://www.loria.fr/~contasss/homeE.html
[3] Femto-St, University of Franche-Comté, Montbéliard, France
aurelien.vasseur@pu-pm.univ-fcomte.fr

Abstract. This paper presents a parallel and fault tolerant version of an incremental learning algorithm for feed-forward neural networks used as function approximators. It has been shown in previous works that our incremental algorithm builds networks of reduced size while providing high quality approximations for real data sets. However, for very large sets, the use of our learning process on a single machine may be quite long and even sometimes impossible, due to memory limitations. The parallel algorithm presented in this paper is usable in any parallel system, and in particular, with large dynamical systems such as clusters and grids in which faults may occur. Finally, the quality and performances (without and with faults) of that algorithm are experimentally evaluated.

Keywords: Neural Networks, Learning algorithms, Parallelism.

Introduction

The work presented in this paper takes place in a multidisciplinary project called *Neurad*, involving physicists[1] and computer scientists[2], whose goal is to enhance the treatment planning of cancerous tumors by external radiotherapy. In our previous works [1,2], we proposed an original approach to solve scientific problems whose accurate modeling and/or analytical description is not directly possible. That method is based on the collaboration of computation codes and neural networks used as universal approximators. Thanks to that method, the *Neurad* software provides a fast and accurate evaluation of radiation doses in any given environment (possibly heterogeneous) for given irradiation parameters. In that context, a new learning algorithm has been designed which provides a network of limited size while giving very accurate results.

[1] IRMA/Crest team of the FEMTO-ST institute.
[2] AND team of the LIFC and Algorille team of the LORIA.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 174–187, 2008.

However, the sequential version of our algorithm is restrained by the use of a single machine at the same time, which does not allow the learning of very large data sets due to memory limitations and the induced important computation times. This is why the design of a parallel version has been planned. Our approach uses domain decomposition to exploit parallelism, so that the initial neural network is decomposed in several sub-networks. In order to ensure a good quality of the global network while preserving good performances, a fine tuning of the overlapping of the sub-domains is performed. Moreover, in order to be usable in any kind of parallel system (parallel machine, cluster or grid), a fault tolerance mechanism is included in our parallel algorithm.

In the following section, a brief state of the art on neural networks is presented. Then, our sequential incremental learning algorithm is detailed in Sect. 2. In Sect. 3, our parallel version is fully described as well as our fault tolerance mechanism. Finally, the presented algorithm is qualitatively and quantitatively evaluated in Sect. 4.

1 State of the Art

Since the first developments of neural networks [3], the major encountered problems lie in their building and learning. Indeed, there are some results proving that a multilayer neural network can be used as a universal approximator [4,5]. However, there is no result about how to build an optimal structure. Many algorithms give good results, as the classical back propagation algorithm [6,7]. Moreover, there exist many optimizations for that kind of algorithms. They concern the structure, as the Square MLP [8] or the HPU [9] designs, and the learning process, as the QuickProp [10] or the Rprop algorithm [11]. Nonetheless, they work on static structures which have to be inferred manually according to the user's experience.

In order to solve that recurrent problem, new learning processes have been proposed which aim at dynamically building the structure of the neural network during the learning process. There are two main kinds of such algorithms.

The first kind corresponds to the incremental learning algorithms. Their principle is to begin the learning process with a neural network of minimal size and to progressively increase the number of neurons until satisfying the desired criterion. The addition of a new node is conditioned by the stabilization of the learning process while the requested accuracy is not reached yet. There exist many variants of that incremental process [12,13,14,15].

The second kind of dynamic learning algorithms consists of the symmetric approach, i.e, a decimation process. In this case, the learning process begins with an over-sized complete structure containing a maximal or sufficiently large number of fully connected neurons. Then, during the learning process, the links and neurons which reveal to be useless are deleted. The most known algorithms in this class are probably the "Optimal Brain Damage" algorithm [16] and the "Optimal Brain Surgeon" algorithm [17].

Nevertheless, all those algorithms are sequential, which limit their use to a single mono-processor machine. So, even if they give very good results, they

cannot be used in practice to process very large data sets. This is why there has also been an important effort led towards the parallelization of existing algorithms or the design of specific parallel learning algorithms. J.Torresen and S.Tomita present a detailed report on parallel approaches in the context of classification neural networks [18]. The major approach in this field is to decompose the initial data set and to build several sub-networks. In some cases, an additional global network is used to retrieve the sub-network corresponding to a given input, such as in [19]. However, to the best of our knowledge, all those studies are focused on classification networks and not on approximator ones. Moreover, they do not take into account the robustness of the algorithm when it is used in dynamical parallel systems, in which network or processor faults may occur. This is an important feature of the parallel learning algorithm presented in this paper.

2 Sequential Building/Learning Algorithm

2.1 Network Structure

As mentioned above, it has already been shown that a multilayer neural network can be used as a universal approximator. We use here the common architecture which consists in three layers of neurons (input, hidden and output). The number of neurons in the input layer is determined by the number of parameters of the function to approximate. In the same way, the number of neurons in the output layer is directly induced by the number of outputs of the target function. In the context of the *Neurad* project, which aims at evaluating radiation doses, the number of neurons in the output layer is reduced to a single neuron which delivers the dose. Finally, the last important parameter in the network structure setup is the number of neurons in the hidden layer. As that number does not directly depend on the number of inputs and outputs of the problem, there is no precise rule to compute it. The only external information which may help to fix that number is the variation degree of the input data. However, there is no accurate relation between those two values. It is thus necessary to dynamically set that number of hidden neurons during the learning process to obtain the most suited networks. That incremental building is described in Sect. 2.2.

In addition to the three-layer organization, we have used a HPU (Higher-order Processing Unit) structure [20] in order to enhance the capacity of the network to approximate high degree functions with sharp variations while preserving a limited number of neurons. That structure also permits to obtain faster trainings. It consists in artificially increasing the number of inputs of the network with polynomial combinations of the original inputs up to a maximal degree (referred to as the order of the network). For example, the inputs of an HPU network of order 3 corresponding to an original network with two inputs (x_1, x_2) are $(x_1, x_2, x_1^2, x_1x_2, x_2^2, x_1^3, x_1^2x_2, x_1x_2^2, x_2^3)$.

In our applicative context, we have also noticed that another structural modification that can enhance the results of the neural network is to replace the linear output neurons by sigmoid ones.

2.2 Incremental Building/Learning

The classical back-propagation learning method is known to be rather slow. In order to speed up the training process, we have chosen the Resilient back Propagation (Rprop) algorithm [11], which is one of the most efficient optimizations of that process (See [21] for a complete survey). Its main difference with the classical back-propagation is that it only uses the sign of the error derivative to update the weights in the network. Moreover, the updatings are performed, for each weight, with a distinct value independent from the error. Those values are respectively increased or decreased, similarly to an acceleration or a deceleration, according to the direction of the error evolution.

Concerning the incremental building of the hidden layer of the network, the principle of our algorithm, depicted in Fig. 1, is to perform a Rprop learning over the current HPU neural network until the error either reaches the required accuracy or does not sensibly evolve anymore according to a given threshold.

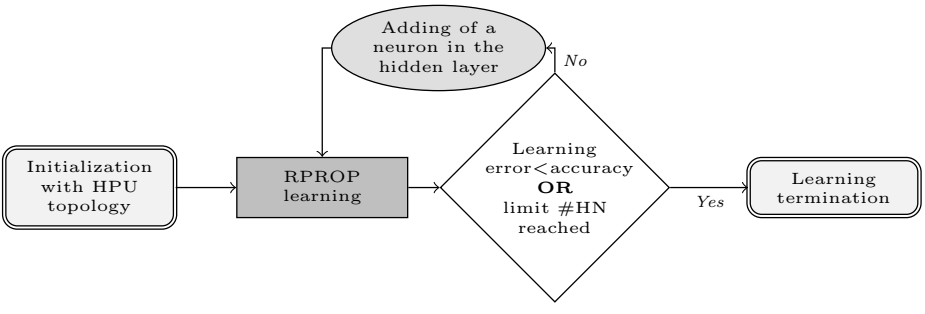


Fig. 1. Incremental building rule

More precisely, our algorithm starts with a given number of hidden neurons (one or a few). Then, when the neural network reaches the desired accuracy, the learning process stops. Otherwise, when the learning limit of the current neural network is considered to be reached (stabilization or over-learning), a neuron initialized with null weights and threshold is added to the hidden layer without modifying the other neurons and links. The null initialization of the additional neuron and the non modification of the other elements in the network are important to avoid any deviation of the current network from its optimization path. After that, the learning process is resumed with that new network configuration. That incremental process is repeated until the desired accuracy is reached or the difference between the results of two consecutive configurations of network becomes too small. That last case corresponds to situations where the overall limit of the network has been reached and the addition of hidden neurons does not improve the results anymore.

As in other learning algorithms, the specification of a validation data set is possible in order to control the learning process and avoid over-learning. Moreover, an upper bound to the number of hidden neurons (limit #HN in Fig. 1) can be specified in order to limit the final size of the network.

Finally, we obtain a sequential and incremental learning process that allows us to build and train efficient and accurate neural networks of limited size for function approximation. Moreover, our approach, used in the particular context of the *Neurad* project, is general and can be used for any kind of data, real or synthetic, and with any kind of activation function of the neurons.

3 Parallel Algorithm

Our parallel algorithm is based on the classical client-server model applied to the distribution of the work. The role of the server is to distribute the different tasks constituting the overall process to the other nodes, which are the clients.

In order to obtain such a simple operating scheme, it is necessary to divide the learning in separate tasks. This is possible according to the principle that the approximation of a function on a given domain can be obtained by performing multiple approximations of that function on sub-domains forming a partition of the initial domain. The following paragraph describes the domain decomposition technique we have used in our algorithm.

3.1 Domain Decomposition

In the case of neural networks, the domain decomposition leads to a composition of several sub-networks in order to perform the overall approximation of the target function. Indeed, the initial domain of the data set is divided into subspaces along one or several of its input dimensions. Obviously, the output dimensions cannot be divided as it would not be possible to know in advance what output sub-domain corresponds to a given input vector.

So, the overall approximation of the initial data set is obtained by the separate learnings of the data subsets induced by the domain decomposition. The decomposition along each dimension can be performed in any way. The simplest decomposition is probably the one which produces data subsets of approximately the same sizes. The decomposition principle is illustrated in Fig. 2.

In addition to the possibility to design a simple and efficient parallel algorithm, the domain decomposition presents another important advantage in the case of

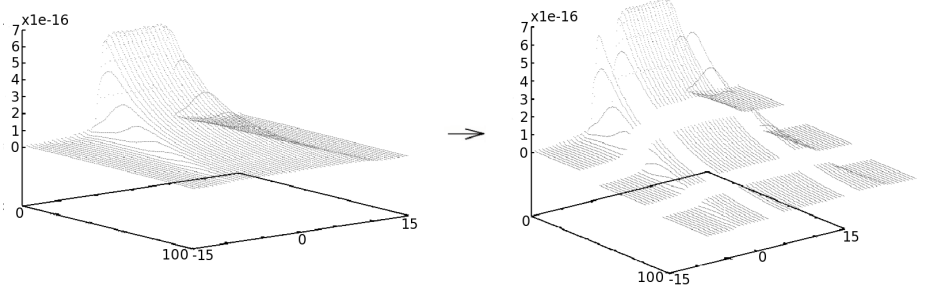


Fig. 2. Domain decomposition in 9 sub-domains of a two-dimensional data set

neural network approximation. It significantly reduces the complexity of the target functions to approximate. Indeed, it is far more easier to approximate a function on a small interval than on a large one, especially if there are sharp variations. Thus, a better accuracy can even be expected on each sub-domain.

However, performing the learnings on sub-domains constituting a partition of the initial domain is not satisfying according to the quality of the results. This comes from the fact that the accuracy of the approximation performed by a neural network is not constant over the learned domain. Thus, it is necessary to use an overlapping of the sub-domains as explained below.

3.2 Overlapping between Sub-domains

As mentioned above, the disadvantage brought by the use of several sub-networks of neurons in place of a single one comes from accuracy problems at the frontiers between each sub-network. Although the neural networks have the capacity of generalization on any given training domain, they do not provide representative results outside this domain and the approximation error increases toward the limits of the domain. This mainly comes from the fact that on the borders of the domain there is less available information about the target function than in the middle of the domain. So, the error is smaller in the middle of the domain than on its borders.

If this is not relevant when using a single neural network, it becomes an issue when several sub-networks are used to represent the domain. Indeed, an increase of the number of sub-networks directly increases the number of frontiers between the sub-domains and then, the number of higher error areas in the domain. Consequently, the average accuracy of the approximation may be importantly reduced. Moreover, the error distribution becomes decomposition-dependent, which is a very restrictive feature as it implies that no decomposition should be made in the areas of higher interest.

Fortunately, there is a solution to that problem which consists in masking the borders of the domains by performing an overlapping of the sub-domains during the learning phase. Thus, we obtain a set of sub-networks whose approximation errors at the frontiers between them is of the same order as anywhere else in the domain. This mechanism implies the distinction, for each sub-network, between its learning domain and its exploitation domain. The former is the domain used to perform the learning of the sub-network; it overlaps with the learning domains of the neighboring sub-networks. The latter is the domain of validity of the sub-network during the exploitation phase; as it is used to find the most suited sub-network to process a given input vector, it does not overlap with the exploitation domain of any other sub-network. The overall principle is depicted in Fig. 3.

In this way, each sub-network has an exploitation domain smaller than its training domain and the accuracy problems at the borders are no longer relevant. Nonetheless, in order to preserve the performances of the parallel algorithm, it is important to carefully set the overlapping ratio α. It must be large enough to avoid the borders errors, and as small as possible to limit the size increase of the data subsets. The trade-off value depends on the nature of the initial data

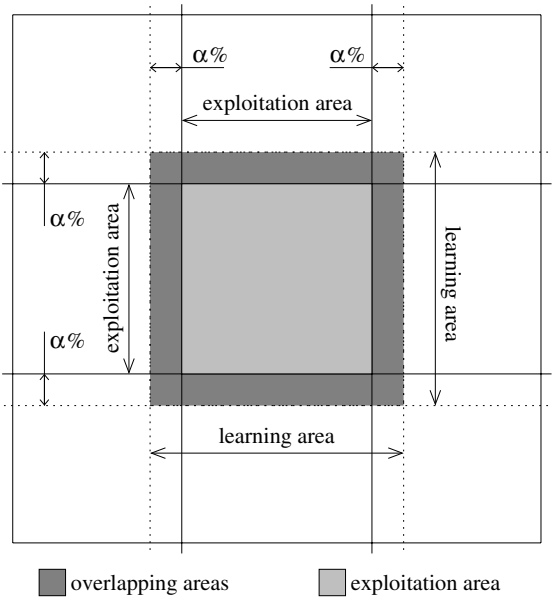


Fig. 3. Overlapping for a sub-network in a two-dimensional domain with ratio α

set and thus, on the application field. In Sect. 4, a case study of that ratio is performed in the context of the *Neurad* project, for radiation dose distributions.

3.3 Fault Tolerance Mechanism

In addition to our parallel scheme, a fault tolerance mechanism is included that enables its use in any dynamical parallel system such as open multiuser clusters or grids. Typically in such contexts, the communication links may be temporarily or definitely interrupted, the processors speed may sharply vary due to the multiuser context and the processors may even stop working. However, it is reasonable to assume that the conjunction of all those possible faults is quite a rare event. Moreover, extreme cases, in which all the processors or links are faulty, are obviously out of consideration. Consequently, we assume in the following that during the learning process, there is always at least one operational node (the server is also the client). With those hypothesis, the goal of our fault tolerance mechanism is to ensure that the learning process continuously progresses as efficiently as possible whatever the events occurring on the system are.

The principle of our system is based on regular message exchanges between the server and the nodes in order to monitor their current state.

The server initializes, for each client, a structure containing:

- its current state: either waiting, learning or in fault
- the identifier of the sub-network it is in charge of
- the date of the last received message from that client

Then, it enters the main learning loop in which it distributes the sub-networks to train to the available clients and monitor their states in order to react to potential faults. Hence, when a client is idle, the server sends it one of the remaining sub-networks to process and its associated data subset.

It is important to note that the sub-network sent may be already partially trained. This is the case when the sub-network has been recovered from a previous learning interruption due to a fault in the system. When the client is processing a sub-network, the server regularly verifies that it is still alive. Finally, in order to avoid the restarting from scratch of a previously faulted learning, the client regularly sends to the server the current version of its sub-network. So, when a fault occurs on the client, the sub-network can be redistributed to another idle client. However, it may occur that a fault of a client comes from a temporary interruption of its link with the server. In such cases, there is no way to identify the cause of the fault and the server will also redistribute the work of that client to another one. Then, when the faulty link comes back working, the server will detect a dual processing of the corresponding sub-network on two clients.

The chosen policy in this case is to let both the clients processing the same sub-network and, as soon as one of them returns the result, the server sends an interruption message to the other client in order to make it accept another work. When there is no idle client although there remains sub-networks to process, the server waits until one of the clients sends back its resulting sub-network after the learning completion. Finally, the main process stops when there is no more sub-network to train and all the clients have returned their results. Once this is made, the server sends a message to all the clients, indicating the end of the overall process, and saves the complete structure of the global network.

On the client side are the complementary operations to the server. When a client receives a training message, it takes delivery of the sub-network and the associated data subset. Then, the node performs the learning of the sub-network using the learning algorithm previously described. During this stage, the node regularly notifies its state to the server by sending the evaluation of its training error. It also sends, less frequently, a safeguard copy of its current sub-network.

Figures 4 and 5 respectively depict the general algorithmic schemes used on the server and on the client sides.

4 Experimental Results

In this section, the quality, performance and robustness of our algorithm are experimentally evaluated. Our algorithm has been implemented in standard C++ with the LAM/MPI communication library [22]. Although that library is not the most suited to fault tolerance, it offers the minimal features of robustness allowing us to validate our algorithm. It must be pointed out that our algorithm does not depend on any implementation environment and another communication library may be used.

The presented experiments have been performed on a multiuser cluster of 20 nodes (Intel PIV, 3Ghz, 1 Go RAM, Debian Linux).

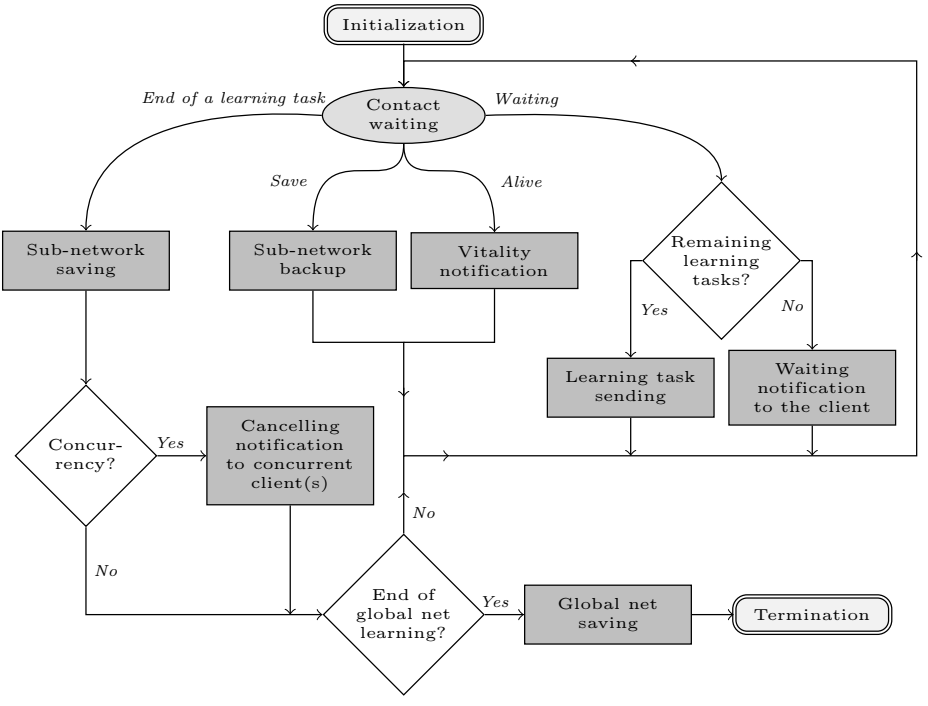


Fig. 4. Server side algorithm

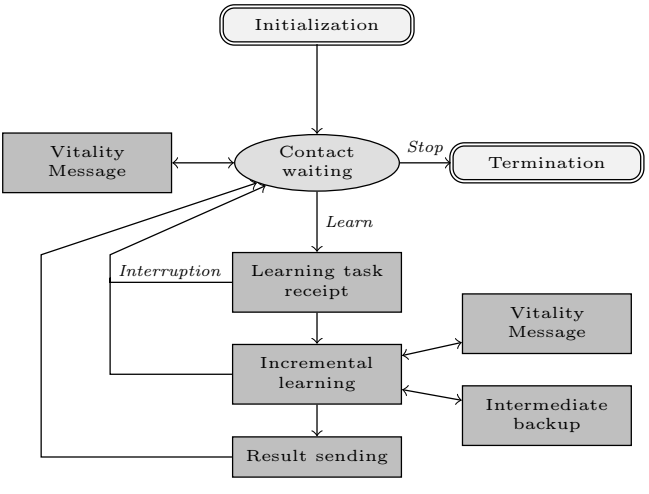


Fig. 5. Client side algorithm

4.1 Quality of the Parallel Learnings

The first test shows the impact of the overlapping between sub-domains on the learning error. We have used a training set containing only one distribution of radiation dose deposit in a homogeneous material to perform this test. The results of that first experiment are presented in Table 1.

Table 1. Impact of the overlapping between sub-domains onto the learning error

Overlapping (%)	0	5	7	10	50
Mean Error (%)	2.86	2.38	2.37	2.37	2.45
Mean Bias (%)	2.28	1.86	1.82	1.83	1.85

This experiment shows that the overlapping is necessary to obtain the best accuracy with our parallel learning algorithm. Effectively, it can be seen that the overlapping ratio directly influences the accuracy of the final network. However, it also shows that this overlapping ratio cannot be fixed a priori without a preliminary study. Indeed, If it is too small, the errors at the frontiers reduce the quality of the final results, and if it is too large, the quality will also be degraded due to a larger size of the sub-domains, implying a potentially higher complexity of the sub-functions to approximate. Those two aspects clearly show that the optimal value of the overlapping ratio is strictly positive but is also far smaller than the maximal possible range. This comes from the fact that the accuracy of the approximation at any given position in the domain, and in particular at the frontiers between sub-domains, directly depends on the number of neighbors used to perform the approximation and, in some sense, on the relative flexibility of the elementary functions used in the neural network. Although there is no analytical way to compute the optimal value of that overlapping ratio for the moment - this will be the subject of future works - it seems quite obvious that this ratio is directly linked to the complexity (maximal and/or mean frequency, value range,...) of the function to approximate.

4.2 Performances of the Parallel Learning Scheme

That second experiment aims at evaluating the impact of the domain decomposition on the performances and quality of our parallel learning algorithm. It has been achieved with a training set generated by the BeamNrc code. That code is a simulator based on the Monte Carlo technique for nuclear applications. The data set is the results of three irradiations of a homogeneous environment of water at three different distances (98, 100, and 102 cm). The set is composed of 1,500,000 points. For each point, we store: the spatial position, the material density and the length between the water environment and the particle accelerator.

The results of our parallel learning algorithm on that data set are presented in Table 2 for several configurations of domain decomposition performed on the three spatial input parameters. The convergence ratio indicates the percentage of sub-networks which have actually reached the requested accuracy. In fact, for

Table 2. Results of our parallel algorithm for several domain decompositions performed on the three spatial dimensions of the training set

Decomposition	$1 \times 1 \times 1$	$2 \times 1 \times 2$	$2 \times 2 \times 2$	$3 \times 1 \times 3$	$3 \times 2 \times 3$	$3 \times 3 \times 3$
Mean Error	6.20e-4	1.57e-4	1.0e-4	1.63e-4	1.0e-4	1.01e-4
Min Error	6.20e-4	9.99e-5	9.99e-5	9.99e-5	9.99e-5	9.99e-5
Max Error	6.20e-4	2.3e-4	1.01e-4	4.97 e-4	1.01e-4	1.23e-4
Convergence ratio (%)	0	25	33	62	66	92
Min Time	4H34	4H06	0H54	1H11	0H04	0H03
Max Time	4H34	8H10	3H25	5H59	3H47	1H42

each sub-network, the learning process may either stabilize before reaching the desired accuracy or not reach it in reasonable time.

The results show that our parallel learning algorithm increases the global accuracy of the neural network while decreasing its learning time. That double gain is due, as mentioned in Sect. 3.1, to the fact the learnings are performed on smaller domains than the initial one.

Concerning the quality of the network, it can be seen that the decomposition of all the input dimensions facilitate the overall convergence of the learning process. Moreover, the convergence rate is also greatly improved according to the number of decomposed input dimensions and the total number of sub-networks.

Concerning the performances, it must be noticed that as the initial training set is quite large, the first result with no domain decomposition has a null level of convergence and its learning time does not actually correspond to the time required to obtain the desired accuracy. So, that time should be far larger.

The larger maximal time of the $3 \times 1 \times 3$ decomposition according to the $2 \times 2 \times 2$ one is due to the difference in complexity of the function to approximate in the respective sub-domains obtained. Moreover, as for the case without any decomposition, the maximal times variations also come from the fact that the learning process may be stopped before the convergence, in order to obtain reasonable times. So, $3 \times 1 \times 3$ has a larger maximal time but also has a far better convergence rate. If the stopping criteria had been based only on the convergence, it is strongly probable that the maximal time of $2 \times 2 \times 2$ would have been larger. It is also the case for $3 \times 2 \times 3$ compared to $2 \times 2 \times 2$.

It has to be noticed that our current version of the algorithm realizes an implicit form of load-balancing by the use of a tasks queue managed in the client-server scheme. However, that load-balancing could be improved - this is also a future work - by performing a non-regular decomposition of the domain in order to obtain approximately the same complexities of the sub-domains. This should induce similar learning times of the sub-domains and thus sensibly enhances the overall learning time of the entire domain.

4.3 Performances of the Robustness Mechanism

As our fault tolerance mechanism allows our algorithm to always successfully terminate as long as there remains at least one operational client node in the system, we focus here on the performances of our algorithm in presence of faults.

However, as such an evaluation should require an entire study in itself, we do not try to be exhaustive here but just give some hints on the behavior of our algorithm. Hence, we use only one frequency of the intermediate backups of the sub-networks here (every 5 learning iterations) but its optimal implementation and value should be discussed in a further study.

We give on the left side of Table 3 the total times of a learning decomposed in 9 sub-domains started with 9 clients in function of the number of permanent client faults occurring during the process. Those results are means of several executions with a general (uniform) random distribution of the faults during the process. In order to make a comparison, the learning times obtained without any fault are given for different numbers of clients on the right side of Table 3.

Table 3. Learning times with 9 initial clients in function of the number of permanent client faults on the left. Learning times without faults in function of the number of clients on the right.

# faults	1	2	3	4	5	6	7	8
Times (min)	13	14	18	19	21	23	26	42

# clients	9	8	7	6	5	4	3	2	1
Times (min)	10	10	11	12	16	16	29	32	58

It can be seen that the progression of the learning times according to the number of faults merely follows the times obtained without faults when the number of clients decreases. However, it is interesting to see that this progression tends to slow down when the number of faults increases.

In fact, those results can be explained by the fact that, additionally to the number of faults, the instants at which the faults occur also have an impact on the performances. The latter they occur during the process, the better are the performances. This comes from two factors. The most obvious and general one is that the latter the faults occur, the longer the algorithm works with a larger number of clients. The latter is more specific to the current version of our parallel learning algorithm as there may be sensible differences between the learning times of the sub-domains. Hence, for the latest faults, it is highly probable that some of the sub-domains learnings are already completed and that some clients are idle, offering the possibility to perform an immediate re-assignation of a stopped learning when a fault occurs. In such cases, the impact of the fault on the performances is minimal.

So, for uniform faults distributions and large numbers of faults during the process, it is highly probable that some of them occur in the particular context described above and thus have a very small impact on the overall performances.

Conclusion

A parallel learning algorithm has been presented which includes fault tolerance. Its principle is based on a domain decomposition of the input parameters of the training data set and on an overlapping of the sub-networks to ensure a good

accuracy of the network on the entire domain of the data set. Moreover, the fault tolerance enables the use of that algorithm on a large class of parallel systems, including dynamical ones.

Qualitative and quantitative evaluations of the algorithm have been performed experimentally on real data sets. They confirm the good behavior of our algorithm in terms of performances, quality and robustness.

In the following of the *Neurad* project, it should be interesting to add another important feature to our learning process which is the possibility to make a network learn new data without loosing its previously accumulated knowledge.

Acknowledgments

The authors thank the LCC (Ligue Contre le Cancer) for the financial support, the OPRAD project (Région Franche-Comté) and the CAPM (Communauté d'Agglomération du Pays de Montbéliard).

References

1. Sauget, M., Martin, E., Gschwind, R., Makovicka, L., Contassot-Vivier, S., Bahi, J.: Développement d'un code de calcul dosimétrique basé sur les réseaux artificiels de neurones. In: 44iémes Journées Scientifiques de la Société Française de Physique Médicale, Avignon, France (June 2005)
2. Bahi, J., Contassot-Vivier, S., Makovicka, L., Martin, E., Sauget, M.: Neural network based algorithm for radiation dose evaluation in heterogeneous environments. In: Kollias, S.D., Stafylopatis, A., Duch, W., Oja, E. (eds.) ICANN 2006. LNCS, vol. 4132, pp. 777–787. Springer, Heidelberg (2006)
3. Pitts, W., McCulloch, W.S.: A logical calculus of ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133 (1943)
4. Cybenko, G.: Approximations by superpositions of sigmoidal functions. Mathematics of Control, Signals, and Systems 2, 303–314 (1989)
5. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Network 2(5), 359–366 (1989)
6. le Cun, Y.: Modéles connexionnistes de l'apprentissage. PhD thesis, Université Pierre et Marie Curie (1987)
7. Rumelhart, D., McClelland, J.: PDP Research group: Parallel Distributed Processing, vol. 1-2. The MIT Press, Cambridge (1986)
8. Flake, G.: Square unit augmented, radially extended, multilayer perceptrons (1998)
9. Giles, C., Taxwell, T.: Learning, invariance and generalization in high-order neural networks. Optical Neural Networks, 344–350 (1994)
10. Fahlman, S.E.: Faster-learning variations on back-propagation: An empirical study. In: Connectionist Models Summer School. Morgan-Kaufmann, San Francisco (1988)
11. Riedmiller, M., Braun, H.: A direct adaptative method for faster backpropagation learning: The Rprop algorithm. In: Proceedings of the IEEE Internationnal Conference on Neural Networks (ICNN 1993), San Francisco (April 1993)
12. Polikar, R., Udpa, L., Udpa, S., Honavar, V.: Learn++: An incremental learning algorithm for supervised neural networks. IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and Reviews 31(4), 497–508 (2001)

13. Dunkin, N., Shawe-Taylor, J., Koiran, P.: A new incremental learning technique. In: Verlag, S. (ed.) Neural Nets Wirn Vietri 1996. Proceedings of the 8th Italian Workshop on Neural Nets, pp. 112–118 (1997)
14. Kurkova, V., Beliczynski, B.: An incremental learning algorithm for gaussian radial-basis-function approximation. In: Second International Symposium on Methods and Models in Automation and Robotics, pp. 675–680 (1995)
15. Fahlman, S.E., Lebiere, C.: The cascade-correlation learning architecture. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems. Denver 1989, vol. 2, pp. 524–532. Morgan Kaufmann, San Mateo (1990)
16. LeCun, Y., Denker, J., Solla, S., Howard, R.E., Jackel, L.D.: Optimal brain damage. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems II. Morgan Kauffman, San Mateo (1990)
17. Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: Optimal brain surgeon. In: Hanson, S.J., Cowan, J.D., Giles, C.L. (eds.) Advances in Neural Information Processing Systems, vol. 5, pp. 164–171. Morgan Kaufmann, San Mateo (1993)
18. Torresen, J., Tomita, S.: A review of parallel implementations of backpropagation neural networks (1998)
19. Guan, S.-U., Qi Yinan, S.K.T., Li, S.: Output partitioning of neural networks. In: Science, E. (ed.) Neurocomputing, vol. 68, pp. 38–53 (October 2005)
20. Ghosh, J., Shin, Y.: Efficient higher-order neural networks for classification and function approximation. Int. J. Neural Systems 3(4), 323–350 (1992)
21. Tertois, S.: Réduction des effets des non-linéarités dans une modulation multiporteuse á l'aide de réseaux de neurones. PhD thesis, Rennes 1 (2003)
22. Burns, G., Daoud, R., Vaigl, J.: LAM: An Open Cluster Environment for MPI. In: Proceedings of Supercomputing Symposium, pp. 379–386 (1994)

High-Performance Query Processing of a Real-World OLAP Database with ParGRES

Melissa Paes[1,2], Alexandre A.B. Lima[3], Patrick Valduriez[4], and Marta Mattoso[1]

[1] COPPE, Federal University of Rio de Janeiro, Brazil
[2] Brazilian Institute of Geography and Statistics, Brazil
[3] UNIGRANRIO, Brazil
[4] INRIA and LINA, University of Nantes, France
melissa@cos.ufrj.br, abento@unigranrio.com.br,
Patrick.Valduriez@inria.fr, marta@cos.ufrj.br

Abstract. Typical OLAP queries take a long time to be processed so speeding up the execution of each single query is imperative to decision making. ParGRES is an open-source database cluster middleware for high performance OLAP query processing. By exploiting intra-query parallelism on PC clusters, ParGRES has shown excellent performance using the TPC-H benchmark. In this paper, we evaluate ParGRES on a real-world OLAP database. Through adaptive virtual partitioning of the database, ParGRES yields linear and very often super-linear speedup for frequent queries. This shows that ParGRES is a very cost-effective solution for OLAP query processing in real settings.

1 Introduction

A Data Warehouse (DW) is a large database repository that integrates data from different sources and provides a single view of all or part of the business data collection in order to improve data analysis. Online analytical processing (OLAP) is a multidimensional approach to analyze data and is used much in DW. This analysis is done through complex queries with high processing costs. The processing of a sequence of such queries can take hours or days. Often, this time-frame is above the deadline the decision makers have to obtain the required analysis.

One of the challenges for OLAP applications is the reduction of individual query processing time. To improve performance in these applications, high-performance parallel database systems can be used [22]. Parallel database systems exploit inter- and intra-query parallelism. Inter-query parallelism runs each query sequentially, but many queries in parallel, thus improving overall throughput. Inter-query aims at OLTP queries running as many concurrent queries in parallel as possible. However, in OLAP, intra-query parallel processing is mandatory. There is an explicit order and sequential dependency between related queries. Running these queries in parallel will not lower the long running time of each query. The problem of using parallel database systems for high performance query processing is the vendor dependency and costs involved in software, hardware, and physical database design.

The database cluster approach [1, 3, 8, 9] improves the performance of queries by applying parallel processing on top of a cluster of databases. High-performance

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 188–200, 2008.

comes with a low cost implementation on top of PC clusters. Akal *et al.* [1] define a database cluster (DBC) as a standard computer cluster (a cluster of PC nodes) running a Database Management System (DBMS) instance at each node. A DBC middleware intercepts queries from an application and manages query execution by taking advantage of the DBC.

Following the DBC approach, ParGRES [9, 12] has been developed as a middleware to provide for intra-query parallel processing on DBC. ParGRES is an open-source middleware between the application and DBMS that intercepts SQL queries to exploit intra-query parallelism. It orchestrates its execution with the help of the sequential DBMS, each running at a cluster node. ParGRES can also provide for inter-query parallelism. To preserve the sequential DBMS execution, virtual partitioning on a replicated database has been proposed in [1] to achieve intra-query parallelism. ParGRES builds upon the virtual partitioning technique by creating an adaptive virtual partitioning that also allows for load-balancing. These features make ParGRES a unique solution for OLAP queries among several DBC systems, like PowerDB [14], C-JDBC [3], and Sequoia [18],

The efficiency of ParGRES has been validated through experiments with typical queries of the TPC-H benchmark [20]. TPC-H represents OLAP business applications and has twenty two heavy-weight select queries and two update queries. Super-linear or linear speed-up was obtained with ParGRES [9] in all queries.

The TPC-H benchmark can be considered a typical, well-behaved application. In our previous work [9] we had to force a non-uniform valued database to test query load-balancing. In this work, we evaluate ParGRES' performance with a real-world OLAP application, from the Brazilian Institute of Geography and Statistics – IBGE. We focus on analyzing the ParGRES non-intrusive approach of its dynamic adaptive virtual partitioning design. This partitioning along with its load balancing during intra-query parallel processing is critical in the high-performance of OLAP queries.

IBGE is the main provider of data and information about Brazil and is the government agency responsible for Brazil's censuses. IBGE provides for *ad-hoc* query access to the census data through the Statistical Multidimensional Database (BME), which is an OLAP database. Such queries are used by several municipalities and Federative Units to define urban planning. BME is also used in emergency situations such as evaluating the effects of a devastating disaster on local populations. BME is a large database and in our tests, we use a 20 GByte database. We analyzed BME query logs for the last two years to come up with fourteen typical SQL queries. These queries have several joins including fact tables as well as dimension tables, aggregations, sub-queries, and predicates with all kinds of ranges of selectivity factors.

To evaluate BME queries, all we had to do was to give ParGRES the names of the fact tables and their partitioning attributes, on which we created clustered indexes[1]. We ran BME queries on ParGRES by using a PC cluster with up to 64 nodes from Grid'5000 [4] and PostgreSQL [13] as the sequential DBMS running on each node. We replicated the BME database to be virtually partitioned during query execution. We performed several experiments with ParGRES and we obtained linear and almost always super-linear speedup on queries frequently issued to the BME census. We

[1] Clustered indexes are typically found in most DBMS. They order (cluster) tuples physically according to the index.

noticed that, by using only a 4 node PC cluster, queries that take about 15 minutes drop to just half a minute. Since queries are *ad-hoc*, we did not perform any fine tuning nor any optimization on PostgreSQL. Application migration costs and investments on this solution are negligible, being restricted to acquiring an off-the-shelf PC cluster. These results can be further improved if caching and other optimizations are used. We believe the same results can be obtained with other OLAP applications such as TPC-H and BME. Particularly in Brazil, the government has many other OLAP databases that could benefit from ParGRES. Extensive results show that ParGRES has proved to be a very cost-effective alternative solution for OLAP query processing in real scenarios.

This paper is organized as follows. Section 2 shows ParGRES' main features. Section 3 describes the BME census database used in our experiments. Section 4 discusses experimental results. Related work is presented in Section 5 and Section 6 concludes.

2 ParGRES

ParGRES [9, 12] is a high-performance database cluster middleware specially developed for OLAP query processing. By exploiting inter- and, in particular, intra-query parallelism, it significantly speeds up the execution of heavy-weight queries, typical from OLAP applications. Fig. 1 shows the general architecture of a DBC with ParGRES.

Before using ParGRES, it is necessary to set the number of nodes to process queries in parallel. When ParGRES receives an SQL query from client applications, it parses and analyzes it in order to determine if it can be processed through intra-query parallelism (following the guidelines specified in [1]). If this is the case, all nodes set in ParGRES are involved in the parallel processing. It re-writes the query in sub-queries to be executed by the DBMS of the database cluster. Sub-queries are generated by using the technique called Virtual Partitioning (VP) [1]. Basically, it consists in rewriting the original query by adding range predicates to it, which originates as many sub-queries as the number of nodes available. Then, each node receives a different sub-query. If the database is fully replicated at all nodes, ParGRES can assign any sub-query to any node. Let us give an example with query Q on table part:

```
Q:      select sum (price)
        from part
        where category = 'Y';
```

In order to implement VP, Q would be rewritten as follows:

```
Qi:     select sum (price)
        from part
        where category = 'Y'
        and pid >= :v1 and pid < :v2;
```

In the example, `pid` is chosen as the *virtual partitioning attribute* (VPA) for table `part`. During query rewriting, a different pair of values (v1, v2) is assigned to each node. Similarly to Akal *et al.* [1], ParGRES does that by equally dividing the total value range currently assigned to `pid` by the number of nodes that are going to process the query and giving each one exactly one sub-range. If, for example, `pid` current range is [1, 1000] and four nodes n_j (j = 1 to 4) are configured in ParGRES, to process Q, node n_1 receives v1 = 1 and v2 = 251, n_2 receives v1 = 251 and v2 = 501, and so on. At each node the local sequential DBMS processes the sub-query on the specified exclusive virtual partition of the table `part`. This way, a database that is not physically partitioned can still be processed through intra-query parallelism. VP is adopted by ParGRES but only to perform its initial query rewriting.

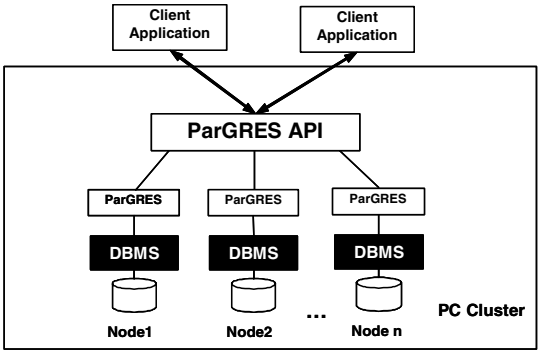


Fig. 1. ParGRES Database Cluster Architecture

In ParGRES, when a node first receives a VP sub-query with its value range to VPA, it does not directly submit it to the local DBMS. At each node, ParGRES locally subdivides the value range generating "smaller" virtual partitions, i.e. sub-queries. The idea is to have more flexibility on the load each node has to process. Since VP is based on table replication any node can process any sub-query. The approach of equally dividing the initial range by the number of nodes relies on the absence of data skew. Otherwise once a DBMS receives its sub-query the DBC can no longer reassign part of it to another node to do load balancing.

At the node level, ParGRES adopts a variation of VP called Adaptive Virtual Partitioning (AVP) proposed by Lima *et al.*[7]. The definition of the value range of the VPA is dynamically generated by adapting it to the processing response time of the nodes. When a node receives its sub-query, it subdivides the predicate value range into smaller ranges. Then it starts processing the range by sequentially submitting as many sub-queries as it takes to the local DBMS. Range sizes are continuously adapted by AVP in order to avoid full table scans (more details in [8]). Such an approach of having many sub-queries per node allied to database replication allows for dynamic load balancing. ParGRES redistributes the load by reassigning virtual partition ranges between nodes during query execution, as described in [8]. When a node finishes processing its sub-query, it sends messages to other nodes offering help to process their sub-queries. Due to replication, a node that needs help can reassign part of its unprocessed attribute range to this idle node.

VP requires an extra query processing phase to compose the final result from the partial ones produced by the sub-queries. ParGRES employs HSQLDB [5], a fast lightweight open-source DBMS, to perform result composition. For example, for processing Q, ParGRES would create a temporary table on HSQLDB as follows:

```
create table tempResult (sprice double);
```

Each sub-query result would then be inserted into `tempResult`. When all nodes are done, ParGRES performs final result composition by issuing the following query to HSQLDB:

```
select sum (sprice) from tempResult;
```

The final result is sent to the client application and `tempResult` is then discarded.

ParGRES is fully distributed and it has local components running on each node. Result composition is a two-phase process: in the first phase, each node uses its local HSQLDB instance to perform local result composition; when all nodes are done, their final results are sent to global ParGRES components that use a global HSQLDB the same way to obtain the final result.

ParGRES also supports data updates, requiring no read-only query processing interruption to process them. As data updates are beyond the scope of this work, we refer the reader to [9], to find more detail.

3 Population and Housing Census 2000 and BME *Ad-Hoc* Queries

IBGE is responsible to support the demands of several different segments of civil society, as well of other governmental institutions at federal, state and municipal levels [2].

Population and Housing Censuses are the main sources of information about the living conditions of the population in each one of the municipalities and localities of Brazil. The censuses produce basic information for the formulation of public policies and for the decision-making processes of private or governmental investments [2]. The data of the survey we used in our tests was collected in the period of August, 1st to November, 30th, 2000, encompassing 5,507 Brazilian municipalities created and installed up to that date. All information from this Census are disseminated and organized in two data collections: Data of the Universe and Data of the Sample, and each data collection in three themes: Housing Units, Persons and Households.

Through the Internet, IBGE makes its survey results available as dynamic pages, downloadable files and online tools to access data. One of these online tools is the Statistical Multidimensional Database - BME.

BME [11] is an OLAP database developed by IBGE and its content is formed by microdata[2] from IBGE's surveys and the metadata associated to these data. This application is intended for researchers who need to generate their own statistical information using microdata and for professionals involved in planning tasks and decision-making processes. BME allows its users to perform *ad-hoc* queries in a database that has more than one billion tuples.

[2] Microdata are nonagregated data about individual objects (persons, housing units, householders, etc.) collected by statistical surveys, that is, the microdata is a record set about one object.

Population and Housing Census 2000 is available in BME and it is one of the greatest surveys in this database, with about 250 millions tuples. It is also the survey with the greatest number of submitted queries by BME users.

The Data of the Sample specifications in BME contains 84 dimension tables, including one time dimension and nine spatial dimensions, and three fact tables: DOMI (housing units data), PESS (persons data) and FAMI (households data). Time dimension refers to the period of the survey and spatial dimensions refer to the Brazilian territorial area.

Fig. 2 shows the relationship between fact tables and time and spatial dimension tables. It presents time dimension (T004), spatial dimensions (G000, G031, G032, G033, G034, G035, G036, G037, G039 and G042) and fact tables (DOMI, PESS and FAMI). Fig. 2 also shows the cardinality of each table. Time and spatial dimensions are related to all fact tables through their foreign keys.

Fig. 3 shows the relationship between fact tables. DOMI is related to the others fact tables, PESS and FAMI, and FAMI is related to PESS, through its primary key.

Fig. 4 illustrates the relationship between DOMI and other dimension tables, excluding the time and the spatial dimensions. There are 24 dimensions, in this case. Their cardinalities are: |M003| = 12, |M075| = 3, |M078| = 5, |M102| = 4, |M103| = 7, |M104| = 4, |M105| = 4, |M106| = 4, |M109| = 8, |M115| = 8, |M116| = 8, |M128| = 6, |M129| = 8, |M208| = 2, |M209| = 12, |M233| = 5, |M270| = 3, |M272| = 10, |M273| = 11, |M274| = 7, |M275| = 11, |M276| = 11, |M277| = 11 and |M278| = 11.

Fig. 5 shows the relationship between FAMI and seven dimension tables, excluding the time and the spatial dimensions. Their cardinalities are: |M290| = 16, |M291| = 17, |M292| = 17, |M293| = 8, |M295| = 13, |M296| = 16, |M297| = 13.

Fig. 6 illustrates the relationship between PESS and other dimension tables, excluding the time and the spatial dimensions. There are 42 dimensions, in this case. Their cardinalities are: |M159| = 7, |M167| = 4, |M298| = 13, |M300| = 2, |M301| = 13, |M302| = 21, |M306| = 144, |M307| = 54, |M308| = 6, |M309| = 7, |M311| = 3, |M314| = 8, |M315| = 5510, |M320| = 3, |M321| = 5, |M322| = 15, |M323| = 11, |M324| = 12, |M325| = 13, |M326| = 5, |M327| = 63, |M330| = 5, |M331| = 7, |M332| = 7, |M333| = 4, |M334| = 513, |M338| = 224, |M341| = 10, |M342| = 7, |M343| = 5, |M348| = 15, |M355| = 3, |M361| = 8, |M4210| = 98, |M4219| = 260, |M4230| = 99, |M4239| = 261, |M4276| = 5605, |M4279| = 232, |M4300| = 21, |M4354| = 94 and |M4511| = 3.

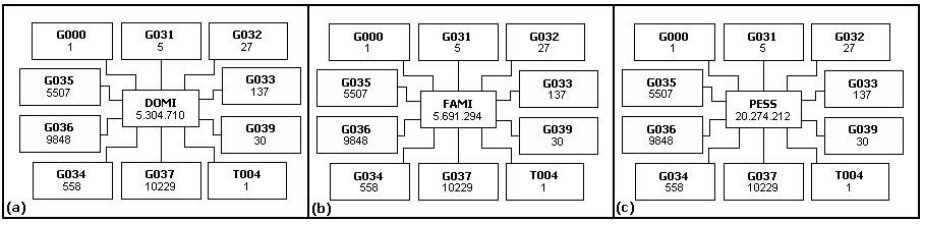


Fig. 2. Relationship between fact tables and dimension tables

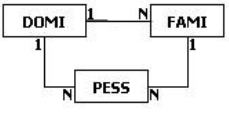


Fig. 3. Relationship between fact tables

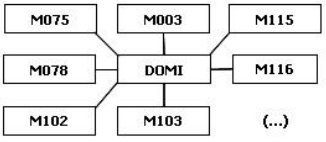


Fig. 4. Relationship between DOMI and dimension tables

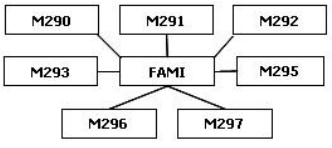


Fig. 5. Relationship between FAMI and dimension tables

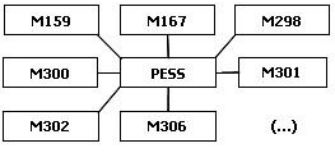


Fig. 6. Relationship between PESS and dimension tables

4 Experimental Evaluation

In this section, we describe our experimental evaluation obtained by running BME OLAP queries on a database cluster managed by ParGRES. Section 4.1 describes our experimental setup and Section 4.2 analyzes the results we obtained.

4.1 Experimental Setup

The experiments were executed on a 64-node cluster of Grid'5000 [4]. Grid'5000 is a French project that provides a large-scale reconfigurable grid infrastructure to support distributed and parallel experiments. Each node has two Intel Xeon 2.3 GHz processors, 4 GB RAM and 160 GB HD. A PostgreSQL 8.2.4 DBMS [13] instance, running on Debian Linux version sid for amd64, is used to manage the Population and Housing Census (BME) database at each node.

The BME database was generated according to specifications from the Statistical Multidimensional application [11]. It contains the dimension and fact tables described in the previous section. We created clustered indexes for the primary keys of each fact table, as required by VP. The total database size is 20 GB and it was fully replicated on the 64 cluster nodes used during our experiments.

Due to *ad-hoc* BME OLAP queries, we analyzed past query loads and selected some typical queries for our experiments. More specifically, we analyze ParGRES performance while executing 14 queries with different complexity levels as shown in Table 1.

All queries use at least one fact table and two out of the dimension tables: one from the time dimension and one from the spatial dimensions. Also, all queries perform at least two aggregations. Queries Q3 and Q6 use only two dimensions (joined to the fact table) and perform four aggregations; Q3 does not have any predicate, while Q6 has a 33.17% selective predicate. Queries Q1, Q2, Q4, Q5, Q7 and Q8 perform joins between a fact table and other dimensions besides the time and the spatial ones. Q1, Q2 and Q4 do not have any predicate. Q4 performs 3 aggregations. Q5 has an 83.95% selective predicate. Q7 and Q8 have high selective predicates, retrieving 2.92% and 3.28% of tuples and perform 4 aggregation functions. Queries Q9, Q10, Q11 and Q13 use two fact tables. Q12 and Q14 use all fact tables in addition to dimension tables. Only Q9 and Q13 do not have predicates, while Q10, Q11, Q12 and Q14 have high selective predicates, retrieving 2.32%, 2.56%, 6.87% and 1.13% of tuples, respectively. Q12 performs 6 aggregations. Q7, Q8, Q10 and Q11 perform a join between a fact table and a relatively large dimension. All queries perform grouping operations. Table 1 summarizes their main characteristics.

Table 1. Main characteristics of each query

Query	Number of aggregations	Predicate	Number of Dimension Tables	Name(s) of Fact Table(s)
Q1	2	N	3	DOMI
Q2	2	N	3	PESS
Q3	4	N	2	PESS
Q4	3	N	3	PESS
Q5	2	Y	3	PESS
Q6	4	Y	2	PESS
Q7, Q8	4	Y	3	PESS
Q9	2	N	4	DOMI, PESS
Q10, Q11	4	Y	4	DOMI, PESS
Q12	6	Y	3	DOMI, PESS, FAMI
Q13	2	N	3	DOMI, PESS
Q14	2	Y	6	DOMI, PESS, FAMI

4.2 Experimental Results

In this section, we present the results obtained during our speedup experiments. Table 2 shows the execution times (in seconds) obtained for each BME query while adding cluster nodes (from 1 to 64 nodes).

Table 2 provides evidence of the excellent results obtained, which are better shown in Fig. 7, that we split in two figures to improve legibility. We present the normalized execution time for each query in Fig. 7 (a) and (b). Normalization was obtained by dividing each query execution time by the highest execution time of its associated query, i.e., the sequential execution of the query. With the sole exception of Q1, which is already quite fast, all other queries have super-linear speedup for all node configurations, as shown in Fig. 8. The speedup curves in Fig. 8 (a) and (b) are shown using log scale for y-axis and the x-axis represents the number of nodes. Fig. 8 also shows the linear speedup curve plotted with the thickest line style.

Table 2. Execution times (in seconds) per number of nodes

Query	1 node	2 nodes	4 nodes	8 nodes	16 nodes	32 nodes	64 nodes
Q1	31.90	13.59	9.00	7.86	2.61	2.50	2.36
Q2	217.41	113.30	24.41	17.85	13.70	7.18	2.50
Q3	217.47	107.60	16.81	13.96	7.94	5.78	2.19
Q4	217.90	101.76	16.89	12.88	12.36	4.79	2.24
Q5	217.44	88.77	10.70	10.38	9.12	4.47	1.69
Q6	217.62	92.48	11.45	11.42	7.44	1.91	1.75
Q7	217.76	85.98	12.33	10.55	5.60	1.51	1.35
Q8	217.67	86.19	10.08	9.01	8.35	2.30	1.81
Q9	460.30	208.19	41.30	22.52	11.03	5.71	3.60
Q10	586.65	244.96	16.18	10.86	8.86	5.70	1.70
Q11	568.54	273.66	13.35	12.74	11.54	2.30	1.82
Q12	1.242.22	618.85	174.95	84.45	42.25	21.91	11.43
Q13	218.37	112.56	23.15	11.68	8.15	5.83	2.94
Q14	1.030.08	511.11	93.80	48.08	25.50	14.45	9.20

With 4 nodes, the virtually partitioned database may be fitting in the four nodes main memory, which explains the significant speedup for all queries with just 2 or 4 nodes in Fig 7. However, even after the memory effect, execution time continues to drop significantly up to the 64 nodes. Q12 is the most time consuming query and its execution drops from 21 minutes to 1.5 minute with 8 nodes and to only 11.43 sec with 64 nodes. Even though some queries present some skew (Q7, Q8, Q10, Q11, Q12 and Q14), it was the same behavior of ParGRES while processing queries with uniform data. Considering that several time-consuming queries are submitted one after the other, often with one depending on the result of the other, intra-query parallelism of ParGRES can significantly reduce the overall time needed for the decision making process. This is crucial in many situations such as disease control measures based on census dimensions and fact tables.

5 Related Work

Besides ParGRES, there are several DBC solutions designed to improve query processing performance through parallel processing techniques, like PowerDB [14], C-JDBC [3], Apuama [10] and Sequoia [18], which is a follow up of C-JDBC. Only ParGRES and PowerDB provide for intra-query parallelism, but PowerDB presents severe limitations in the presence of load unbalancing between nodes during query processing.

PowerDB [14] is a DBC middleware that uses full database replication as physical data organization scheme to obtain parallel query processing. Different middleware solutions have been proposed in PowerDB for OLAP and OLTP query processing. With respect to OLAP, Akal *et al.* [1] have originally proposed VP to obtain intra-query parallelism. However, VP has limitations as it relies on a uniform data value distribution for the VPA to achieve high-performance during query processing. Its "one sub-query per node" approach is also an issue as it prevents dynamic load balancing due to black-box DBMS. In our previous work [9] we show that our dynamic AVP outperforms VP for all TPC-H queries. Particularly, in the presence of data skew, the static features of using just VP imposed poor performance results, often with slow down factors.

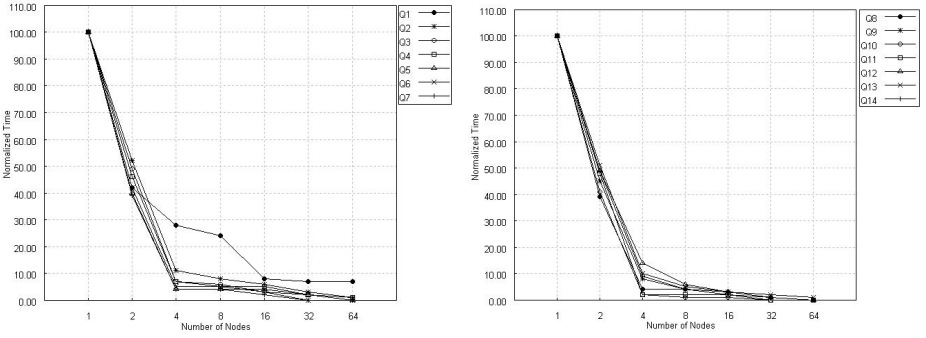


Fig. 7. Query execution time experiments: (a) results for Q1-Q7; (b) results for Q8-Q14

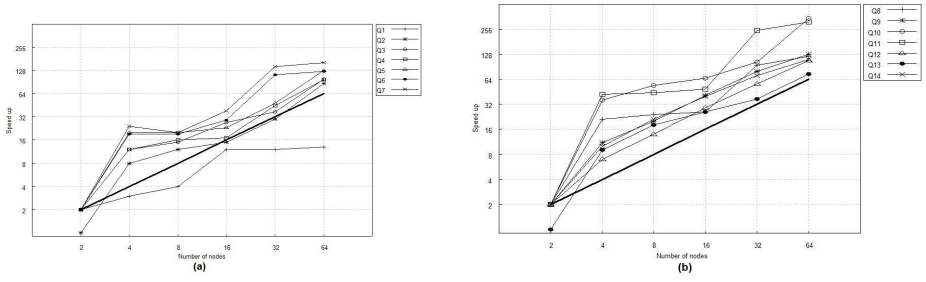


Fig. 8. Query speedup: (a) results for Q1-Q7; (b) results for Q8-Q14

Röhm et al. [15] propose Hybrid Partitioning (HP), a physical database design approach that avoids full database replication by equally fragmenting the fact tables and allocating each fragment to a different cluster node. The dimensions are fully replicated between nodes. HP makes it possible to implement intra-query parallelism during OLAP query processing. However, if load unbalancing occurs during query execution, data transfers must be made in order to redistribute load. In fact, no dynamic load balancing policy is proposed by the authors. Other works were proposed in the context of PowerDB [16, 17] but they do not employ intra-query parallelism thus not reducing the execution time of individual queries, which is imperative for speeding up decision making process. Finally, PowerDB is not open-source nor is available for download.

C-JDBC is an open-source DBC middleware designed by Cecchet *et al.* [3] just as ParGRES. However, it focuses on OLTP and e-commerce applications. It uses full and partial database replication to obtain inter-query parallelism. Intra-query parallelism is not supported. C-JDBC shows good performance for the TPC-W Benchmark [21]. In particular, partial replication is better than full replication because of data updates, typical from e-commerce applications. However, without intra-query parallelism, C-JDBC is not adequate to OLAP query processing. It is more suited to applications that issues fast concurrent queries, such as TPC-C [19] and TPC-W [21].

Sequoia [18] is the sequence of the C-JDBC project and is a solution that offers clustering, load balancing and failover services for any database. Sequoia implements the concept of Redundant Array of Inexpensive Databases (RAIDb) (just as C-JDBC) to provide partial replication in order to tune the degree of replication of each database. It also improves performance using fine grain query caching and transparent connection pooling. Furthermore, Sequoia offers support for clusters of heterogeneous database engines. Just as C-JDBC, it fits better for OLTP and e-commerce applications.

Apuama is proposed by Miranda *et al.* [10] to add intra-query parallelism to C-JDBC. Intra-query parallelism is implemented through static VP and full database replication, thus achieving good results. However, it does not properly deals with the DBMS as a black-box component as it uses DBMS-specific commands to force the use of clustered indexes during query processing in order to have good performance with VP. In addition, as PowerDB, it is also very sensitive to skew, since it cannot do dynamic load-balancing.

6 Conclusions

In this paper, we described an evaluation of ParGRES database cluster middleware in a real-world scenario. We have show experiments with Brazilian official census database (Population and Housing Census 2000), 14 BME OLAP queries using a 64-node cluster of the Grid'5000 experimental platform. The Census 2000 database was generated according to BME specifications and it was fully replicated on the 64 cluster nodes used during our experiments. PostgreSQL DBMS was used to manage the database at each node.

Our experiments explored intra-query parallel processing with time consuming typical BME census queries and the behavior of ParGRES open source database cluster middleware by using a real scenario. The results showed that, in almost all cases, ParGRES yields super-linear speedup while adding cluster nodes (from 1 to 64 nodes). With the exception of Q1 which is already quite fast, all other queries get significant super-linear speedup for all node configurations.

Our experimental results show that AVP improves the performance of *ad-hoc* queries in real scenarios, including data skew. These results are very encouraging and make ParGRES a very cost-effective alternative solution for OLAP applications designed by governmental institutions and researchers in general.

In future work, we will address concurrent sequences of queries. The experiments in this paper have focused on queries in isolation. It is necessary to evaluate the system performance by simulating concurrent queries from a few users. Future experiments should also focus on streams of queries as specified in TPC-H which is also typical of BME decision makers.

Currently ParGRES team is developing GParGRES, its extension to grid platforms. Preliminary results on TPC-H also on Grid´5000 show encouraging results [6]. We also plan to evaluate our BME distributed design on top of grids using larger configurations of nodes. However, dynamic load balancing in grids may compromise the speedup obtained with the cluster.

Acknowledgments

This work was partially funded by CAPES-COFECUB (DAAD project), CNPq-INRIA (GriData project), French ANR Massive Data (Respire project) and the European Strep Grid4All project. The authors are grateful to the Grid´5000 team for their support.

References

1. Akal, F., Böhm, K., Schek, H.J.: OLAP Query Evaluation in a Database Cluster: a Performance Study on Intra-Query Parallelism. In: Manolopoulos, Y., Návrat, P. (eds.) ADBIS 2002. LNCS, vol. 2435, pp. 218–231. Springer, Heidelberg (2002)
2. Brazilian Institute of Geography and Statistics – IBGE, http://www.ibge.gov.br
3. Cecchet, E., Marguerite, J., Zwaenepoel, W.: C-JDBC: Flexible Database Clustering Middleware. In: Proceedings of USENIX Annual Technical Conference, Freenix Track, Boston, EUA, pp. 9–18 (June 2004)
4. Grid5000 Projects Web Site - Grid5000, http://www.grid5000.fr
5. HSQL Database Engine, http://hsqldb.org/
6. Kotowski, N., Lima, A.A., Pacitti, E., Valduriez, P., Mattoso, M.L.Q.: Parallel Query Processing for OLAP in Grids. Concurrency and Computation. Practice & Experience (2008), http://dx.doi.org/10.1002/cpe.1303
7. Lima, A.A.B.: Intra-query parallelism in Database Clusters (in Portuguese). Ph.D. Thesis, COPPE/UFRJ, Rio de Janeiro (2004)
8. Lima, A.A.B., Mattoso, M., Valduriez, P.: Adaptive Virtual Partitioning for OLAP Query Processing in a Database Cluster. In: Proceedings of the 19th Brazilian Symposium on Database Systems (SBBD 2004), Brasília, Brazil, October 18-20, pp. 92–105 (2004)
9. Mattoso, M., Zimbrão, G., Lima, A.A.B.: Baião. F., Braganholo, V., Aveleda, A., Miranda, B., Almentero, B., Costa, M.N.: ParGRES Middleware for Executing OLAP Queries in Parallel. Technical report (2005), http://pargres.nacad.ufrj.br/Documentos/ES-690.pdf
10. Miranda, B., Lima, A.A.B., Valduriez, P., Mattoso, M.: Apuama: Combining Intra-query and Inter-query Parallelism in a Database Cluster. In: Grust, T., Höpfner, H., Illarramendi, A., Jablonski, S., Mesiti, M., Müller, S., Patranjan, P.-L., Sattler, K.-U., Spiliopoulou, M., Wijsen, J. (eds.) EDBT 2006. LNCS, vol. 4254, pp. 649–661. Springer, Heidelberg (2006)
11. Multidimensional Statistics Database – BME, http://www.bme.ibge.gov.br
12. ParGRES, http://pargres.nacad.ufrj.br/
13. PostgreSQL, http://www.postgresql.org
14. POWERDB, http://www.dbs.ethz.ch/archive/index.html
15. Röhm, U., Bohm, K., Schek, H.-J.: OLAP Query Routing and Physical Design in a Database Cluster. In: Zaniolo, C., Grust, T., Scholl, M.H., Lockemann, P.C. (eds.) EDBT 2000. LNCS, vol. 1777, pp. 254–268. Springer, Heidelberg (2000)
16. Röhm, U., Bohm, K., Schek, H.-J.: Cache-Aware Query Routing in a Cluster of Databases. In: Proceedings of the 17th International Conference on Data Engineering (ICDE 2001), pp. 641–650. IEEE Computer Society, Los Alamitos (2001)
17. Röhm, U., Böhm, K., Scheck, H.-J., Schuldt, H.: FAS - A Freshness-Sensitive Coordination Middleware for a Cluster of OLAP Components. In: Proceedings of the 28th International Conference on Very Large Databases Conference (VLDB 2002), Hong Kong, China, August 20-23 (2002)

18. Sequoia Project, http://sequoia.continuent.org/HomePage
19. TPC Benchmark C, http://www.tpc.org/tpcc/
20. TPC Benchmark H, http://www.tpc.org/tpch/
21. TPC Benchmark W, http://www.tpc.org/tpcw/default.asp
22. Valduriez, P.: Parallel Database Systems: open problems and new issues. International Journal on Distributed and Parallel Databases 1(2), 137–165 (1993)

Improving Search Engines Performance on Multithreading Processors

Carolina Bonacic[1], Carlos Garcia[1], Mauricio Marin[2], Manuel Prieto[1], Francisco Tirado[1], and Cesar Vicente[1,⋆]

[1] Depto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid
cbonacic@fis.ucm.es, {garsanca,mpmatias,ptirado}@dacya.ucm.es
[2] Yahoo! Research Latin America
Santiago of Chile
mmarin@yahoo-inc.com

Abstract. In this paper we present strategies and experiments that show how to take advantage of the multi-threading parallelism available in Chip Multithreading (CMP) processors in the context of efficient query processing for search engines. We show that scalable performance can be achieved by letting the search engine go synchronous so that batches of queries can be processed concurrently in a simple but very efficient manner. Furthermore, our results indicate that the multithreading capabilities of modern CMP systems are not fully exploited when the search engine operates on a conventional asynchronous mode due to the moderate thread level parallelism that can be extracted from a single query.

1 Introduction

The algorithmic design and implementation of current Web Search Engines is based on the asynchronous message passing approach to parallel computing in which each newly arriving query is serviced by an independent thread in a classical multiple masters/slaves scheme. Typical facilities for parallel query processing at data centers are composed of a few thousand Linux boxes forming clusters of computers.

On the other hand, the amount of work demanded by the solution of queries follows the so-called Zipf's law which in practice means that some queries, in particular the ones composed of most popular terms, can demand large amounts of processing times whereas others containing less frequent terms can require a comparatively much smaller processing time.

Thus under this asynchronous approach and hardware latencies a given query can easily restrain smaller queries by consuming comparatively larger amounts of resources in processor cycles, disk and inter-processors network bandwidths.

⋆ This work has been partially supported by the research contracts CICYT-TIN 2005/5619, CYTED-506PI0293, FONDECYT 1060776 and Ingenio 2010 Consolider CSD2007-20811. We also thank Sun Microsystems and the AulaSun of the UCM for their support.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 201–213, 2008.

However, we have found that a careful design of the major steps involved in the processing of queries can allow its decomposition in such a way that we can let every query share the cluster resources evenly in a round-robin manner [8,9]. We have observed that this scheme can be particularly useful in preventing unstable behavior under unpredictable variations in the query traffic arriving to the search engine.

In particular, we have observed for the standard asynchronous method of query processing that sudden peaks in the query traffic can be very detrimental to overall performance due to the Zipf's law distribution of the workload per query. We have also observed that the round-robin method of query processing solves this problem efficiently. We have validated this claim through extensive experimentation by running actual query logs upon actual 1TB samples of the Web. Nevertheless, our experiments have been performed on standard machines with coarse-grain threads implemented by Posix software running on clusters supporting the distributed memory model and the MPI message passing communication library.

Having said that, it is clear that this discussion is only valid at a macroscopic level in terms of "heavy" threads and operations for query processing in a sharing nothing model for data distribution. However, state of the art computer architectures integrate facilities for light threads and shared memory, which are available to the programmer in the form of efficient realizations of the *OpenMP* model of parallel computing. In fact, future improvements in processor performance will predominantly come from Thread Level Parallelism, rather than from increasing clock frequency or processor complexity [2]. In this regard, we think that it is an interesting research problem to validate the above claims in the context of these new architectures and if not, explore new optimizations to achieve efficient performance under this new setting.

In this paper, we provide a first step in this direction by studying different realizations of standard and round-robin search engines implemented upon a state-of-the art Chip Multithreading system [11] and its respective *OpenMP* realization. As experimental platform we have chosen a Sun Microsystems' UltraSPARC T1 processor – code-named as Niagara [6] and marketed by Sun as *CoolThreads* technology – since it symbolizes the recent shift to CMP in the server market and presents a radical new approach to enable throughput computing and scalability with low power consumption.

For programming purposes the T1 can be seen as a set of logical processors that share some resources. Consequently, one may think that parallelization schemes targeted for other shared-memory multiprocessors, such as SMP systems, are also good candidates for this processor. However, the sharing of resources introduced on the T1 for increasing utilization may cause serious bottlenecks and hence, strategies that are appropriate for these machines may be inappropriate or less effective for the T1. One of the goals motivating this study is to revise the implementation of parallel search engines in this light.

The rest of this paper is organized as follows: Section 2 and 3 describe our search engine and the experimental framework respectively. Section 4 presents

our parallel proposals and Section 5 shows some performance results. Finally the paper ends with some conclusions and hints for future research.

2 Search Engine Overall Description

2.1 Distributed Inverted File

Web Search Engines use the inverted file data structure to index the text collection and speed up query processing. A number of papers have been published reporting experiments and proposals for efficient parallel query processing upon inverted files which are distributed on a set of P processor-memory pairs [1,3,4,7,8,9,10,14]. It is clear that efficiency on clusters of computers is only achieved by using strategies devised to reduce communication among processors and maintain a reasonable balance of the amount of computation and communication performed by the processors to solve the search queries.

An inverted file is composed of a vocabulary table and a set of posting lists. The vocabulary table contains the set of relevant terms found in the collection. Each of these terms is associated with a posting list which contains the document identifiers where the term appears in the collection along with additional data used for ranking purposes. To solve a query, it is necessary to get the set of documents *ids* associated with the query terms and then perform a ranking of these documents so as to select the top K documents as the query answer.

Current search engines use the document partitioned approach to distributing the inverted file on a set of P processors. In this case, the document collection is evenly distributed at random on the processors and an inverted file is constructed in each processor considering only the documents stored in the processor. Solving a query involves to (a) place a copy of it in each processor, (b) let each processor calculate their local top K results and (c) make a merge of all results to select the global top K results.

Query operations over parallel search engines are usually read-only requests upon the distributed inverted file. This means that one is not concerned with multiple users attempting to write information on the same text collection. All of them are serviced with no regards for consistency problems since no concurrent updates are performed over the data structure. Insertion of new documents is effected off-line.

2.2 Organizing Query Processing

At the parallel server side, queries arrive from a receptionist machine that we call the *broker*. The *broker* machine is in charge of routing the queries to the cluster's processors (where for the scope of this paper each processor is a chip-multiprocessor node of the cluster) and receiving the respective answers. It decides to which processor routing a given query by using a load balancing heuristic. The particular heuristic depends on the approach used to partition the inverted file. Overall the *broker* tends to evenly distribute the queries on all processors.

More in detail, the parallel processing of queries is basically composed of a phase in which it is necessary to fetch parts of all of the posting lists associated

with each term present in the query, and perform a ranking of documents in order to produce the results. After this, additional processing is required to produce the answer to the user. This paper is concerned with the fetching+ranking part. We are interested in situations where it is relevant to optimize the query throughput.

A relevant issue for this paper is the way we organize query processing upon the piece of inverted file stored in each processor. We basically apply the combination of two strategies we have devised to efficiently cope with hardware resource contention among queries and dynamic variations in the query traffic:

- **Round robin query processing.** We let queries to use a fixed quantum of computation, communication and disk access before granting the resources to another query in a round-robin fashion.
- **Operation mode.** We dynamically switch the mode of operation of the search engine between the asynchronous and synchronous message passing modes of parallel computation in accordance with the observed query traffic.

In the following subsection we describe both strategies in detail.

2.3 Iterative Ranking and Round-Robin Query Processing

The processor in which a given query arrives is called the *ranker* for that query since it is in this processor where the associated document ranking is performed. Every query is processed iteratively using two major steps:

- **Fetching.** The first one consists on fetching a K-sized piece of every posting list involved in the query and sending them to the *ranker* processor. In essence, the *ranker* sends a copy of every query to all other P nodes. Next, all nodes send K/P pairs (*doc_id, frequency*) of their posting lists to the *ranker* which performs the first iteration of the documents ranking process.
- **Ranking.** In the second step, the *ranker* performs the actual ranking of documents and, if necessary, it asks for additional K-sized pieces of the posting lists in order to produce the K best ranked documents that are passed to the *broker* as the query results. We use the vectorial method for performing the ranking of documents along with the filtering technique proposed in [12]. Consequently, the posting lists are kept sorted by frequency in descending order. Once the *ranker* for a query receives all the required pieces of posting lists, they are merged into a single list and passed throughout the filters. If it happens that the document with the less frequency in one of the arrived pieces of posting lists passes the filter, then it is necessary to perform a new iteration for this term and all others in the same situation.

Thus the ranking process can take one or more iterations to finish. In every iteration a new piece of K pairs (*doc_id, frequency*) from posting lists are sent to the *ranker* for each term involved in the query. This concept of iteration is essential to distribute and allocate system resources to the queries in a round-robin fashion: the quantum comes from the fact that we let queries work on chunks of posting lists of size K and organize document ranking in iterations.

2.4 Operation Mode

As mentioned above, we dynamically switch the mode of operation in accordance with the query traffic observed.

- **Asynchronous mode.** Low query traffic triggers an asynchronous mode in which each query is serviced by a unique *master* thread in charge of processing the query. This *master* thread can communicate with P other *slave* threads, each located in one of the P cluster nodes.
- **Synchronous mode.** High query traffic triggers a mode in which all active threads are blocked and a single thread takes the control of query processing by grouping queries in batches and processing them sequentially. In this case messages are buffered in all cluster nodes and sent out at the end of the current batch being processed, point at which all processors are barrier synchronized. Better utilization of system resources of this mode comes from the fact that overheads such as thread scheduling and synchronization cost are reduced significantly and communication is performed in bulk.

3 Experimental Framework: Computing Platform and Data Set

As experimental platform, we have chosen a Sun Microsystems' UltraSPARC T1 processor, whose main features are summarized in Table 1. Initially codenamed as *Niagara*, the T1 is a special-purpose CMP designed by Sun for the server market. It is available with four, six or eight CPU cores, and each core allows for the execution of four threads concurrently. Essentially, T1 cores are fine-grain multithreading (FGM) processors [5] that switch between threads of execution on every cycle for hiding the inefficiencies caused by long operational latencies such as memory accesses [13]. Single thread applications will perform better on traditional processors, but multithreaded workloads may benefit from this architecture: each thread is slower but this architecture yields better use of the processor's resources and potentially a higher overall throughput.

In our implementations, thread level parallelism has been exploited by means of the *OpenMP* standard, which is supported by Sun's native compilers.

3.1 Fixed-Point Ranking

The UltraSPARC-T1 processor has a limited floating-point capability since it only provides one floating-point unit to support all 8 cores on the chip, i.e. only one thread can use it at a time. Furthermore, even if just one thread uses the floating-point unit, there is a 40 cycle penalty to access the unit. Most commercial applications have little or no floating-point content so it is not a major handicap. However, in our target application, one of the most costly phases is the *ranker* process, which uses floating-point arithmetic to classify the most relevant documents for a query.

Table 1. Main features of the target computing platform

Processor	SUN UltraSPARC-T1 8 core processor (1.2GHz) (4-way fine-grain multithreading core)	
	L1 Cache (per core)	16+8 KB (instruction+data) 4-way associative, LRU
	L2 Unified Cache	3MB (4Banksx768KB) 12-way associative, pseudo-LRU
Memory	16 GBytes (4x4GBytes) DIMMS 533 MHz DDR2 SDRAM	
Operating System	SunOS 5.10 (Solaris 10) for UltraSparcT1	
Sun C/C++ Compiler v5.8 Switches	-fast -xarch=v9 -xipo=2 Parallelization with OpenMP: -xopenmp=parallel	

To overcome this potential bottleneck, we have modified the *ranker* to avoid floating-point arithmetic. Our implementation uses a 32-bit fixed-point data representation for holding the *appearance frequency* in the posting lists, instead of the conventional floating-point representation, and performs computations using a customized fixed-point library that takes advantage of the T1 integer ALUs – there is an integer ALU per core –. The overhead introduced in the fixed-point version by overflow checking is around 20 to 40%, but the large penalties introduced by floating-point operations (the floating-point *sqrt* takes thousand of cycles) compensate this cost.

Figure 1 illustrates the potential benefits of this optimization. It shows the scalability of a synthetic benchmark that tries to mimic the kind of computations performed by the ranking process – it mixes different arithmetic operations such as divisions, multiplications, logarithms and square roots –. As expected, the performance of the floating-point version is really poor but the fixed-point counterpart scales reasonable well.

3.2 DataSet Inputs

All the results of this work have been obtained using a Chilean Web database sample taken from *www.todocl.cl*. The index structure contains around 1 million Spanish terms – 1.5 GBytes in size –. Queries have been selected randomly from a set of 127.000 queries extracting from *todocl* logs.

4 Parallel Scheme

As mentioned above, the availability of chip multithreading architectures introduces a new scenario in which thread-level parallelism becomes the key for achieving performance. In this regard, a critical issue here is the operation mode of the search engine since it strongly influences the way in which thread level parallelism can be extracted:

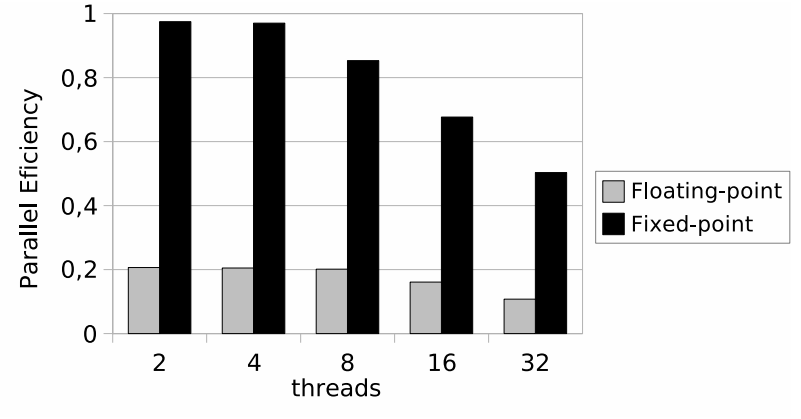


Fig. 1. Parallel efficiency of a synthetic benchmark that tries to mimic the kind of computations performed by the ranking process using either floating-point or fixed point arithmetic

- **Synchronous mode.** Under high query traffic, batching the queries and letting the search engine go synchronous introduces regular coarse-grain parallelism. This parallelism can be easily translated into thread level parallelism in a simple manner by running queries into separate threads. Thread management overheads are relatively small at the expense of synchronization cost. The question here is whether this coarse-grain parallelization fits well with CMP characteristics.
- **Asynchronous mode.** Under low query traffic, it is necessary to resort to other sources of parallelism. The question here is whether intra-query parallelism is high enough to be exploited efficiently on CMP architectures.

In the following subsection we describe both approaches more precisely.

4.1 Synchronous Mode

The coarse-grain parallelism introduced by the synchronous mode can be easily expressed by means of *OpenMP* directives using conventional query distribution schemes. However, the similarities amongst the different threads – they execute the same code with just a different query – may cause contention for the shared resources of the T1, especially cache space and memory bandwidth. We have tried to minimize these penalties with a data distribution and thread assignment strategy that looks for batching queries with similar terms on the same processor. Essentially, our idea is to take advantage of the available temporal locality, to increase the cooperation between threads and avoid costly memory accesses as much as possible.

Algorithm 1 shows the pseudo-code of our parallel approach with a first step done by the *broker* machine (*master*), which tries to gather queries with some terms in common, and then each processor (*slaves*) finds the best documents associated to them in parallel.

Algorithm 1. Our parallel proposal for the synchronous web search engine.

```
At the broker machine do:
// group together queries with similar terms
query_batch = build_clustered_batch(current_queries);
broadcast_to_all_processors(query_batch);

In each processor do:

#pragma omp parallel for private(...) shared(...)
for q=1:Nqueries_processor do

  query = query_batch[q];
  for term=1:terms_in_query do
    posting_list[term] = fetch(term);
  end for
  best_docs[query] = ranking(posting_list, terms_in_query);

end for
...
```

4.2 Asynchronous Mode

For the Asynchronous mode we use an *OpenMP* parallelization of the document ranking routine executed by each query to get the top K documents. This parallelization involves deploying a team of *OpenMP* threads at various points of the routine. In particular, for the cases of identical operations performed over the complete piece of posting list for each query term, we do it in parallel by letting each thread to work on a different segment of the posting list. The filtering technique is a bit more involved. It needs synchronization to update the current score barrier. Further documents down the posting list must beat this barrier in order to be considered as candidates to be included in the top K results. The barrier must be updated concurrently by the threads. This is solved by using a critical section in the points at which this barrier is updated; though this occurs less frequently during the processing of the posting lists.

5 Performance Results

In this Section we attempt to answer those questions raised above. Performance results have been obtained on a single UltraSPARC-T1 processor. This study can be viewed as a first approach of a more complex distributed system based on CMP processors which should behave as *slaves* in our context. Our objective is to analyze in detail the best intra-node parallel approaches and outline some preliminary conclusions which could be extrapolated for a real infrastructure.

Algorithm 2. Our parallel proposal for the asynchronous web search engine.

```
At the broker machine do:
// dispatch queries when they arrive
dispatch_to_processor_P(query);

In each P processor do:

for term=1:terms_in_query do
   #pragma omp parallel private(...) shared(...)
   posting_list[term] = parallel_fetch_and_operations(term);
end for
best_docs = parallel_ranking(posting_list, terms_in_query);
...
```

5.1 Synchronous Mode

Figure 2 shows the throughput achieved with our synchronous proposal. As mentioned above, we have tried to improve implicit cooperation between threads with a data distribution and thread assignment strategy that looks for batching together queries with similar terms. To estimate the potential benefits of this strategy we have compared two extreme scenarios. The gray column corresponds to the most adverse situation: there is no common terms between subsequent queries and all the threads of a given batch compete for the available resources. The black column, in contrast, corresponds to the potentially most favorable situation: all the threads of a given batch process queries with identical terms.

The noticeable difference between both scenarios when running 16 and 32 threads highlights the benefits of a conscious thread distribution. In any case,

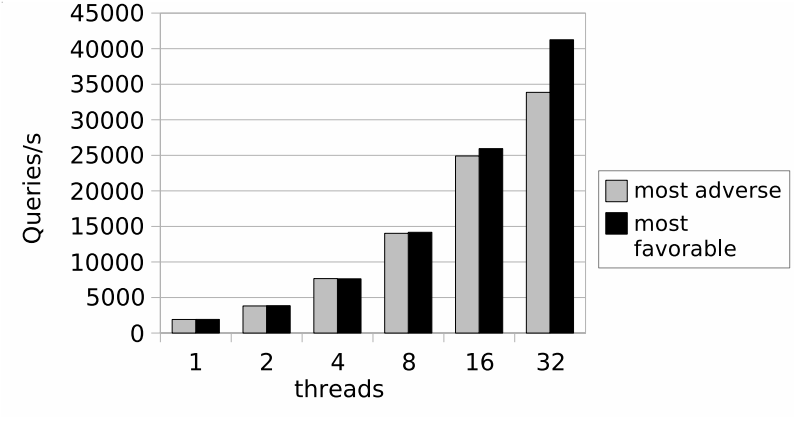


Fig. 2. Query throughput achieved by the synchronous mode in the most adverse (gray column) and favorable (black column) scenarios

the throughput is satisfactory enough in both scenarios. The speedup increases proportionally with the number of threads and in the most favorable scenario, our synchronous search engine reaches a speedup of 22 using 32 threads.

5.2 Asynchronous Mode

Essentially, the experimental results (see Figure 3) show that the gain coming from parallelism is not really significant. This is mainly due to the fact that in each round-robin iteration of the processing of a given query, the amount of data (the ones involved in the pieces of posting lists of size K) that is processed is small. For example, at various points in the document ranking process it is necessary to sort candidate documents. However, trying to do that sorting in parallel by using *OpenMP* threads is not worthwhile since the amount of document to be sorted is not large enough.

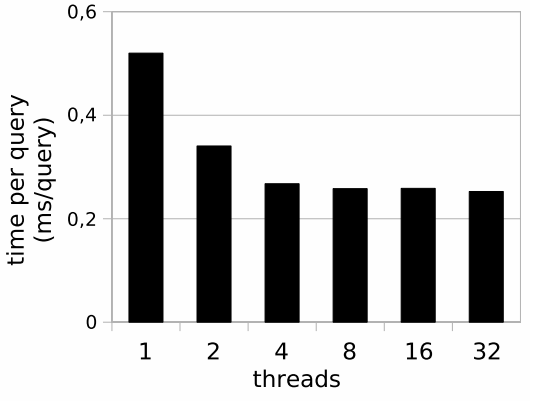


Fig. 3. Time (ms) per query using the asynchronous mode

Recall that round-robin is necessary to prevent large queries from consuming all resources in detriment to small queries. In addition, the filtering technique applied to avoid having to consider the complete posting list for each query term is based on the update of a barrier which finally stop the ranking by deciding that the remaining items in the involved posting lists are not able to include new document among the top K results. When implemented using *OpenMP* threads this barrier become a critical section of the ranking process whose serialization introduces performance degradation.

We should emphasize that current search engines are fully asynchronous and they are prone to this problem as well. In general these machines use techniques to avoid scanning the complete posting list and thus they are essentially in the same difficulties to get advantage of the capabilities provided by CMP systems. In this regard, it can be argued that under high query traffic, the execution of multiple queries would also overlap in the asynchronous mode and this overlapping would provide enough parallelism. In the following we call this ideal case Optimal-Async.

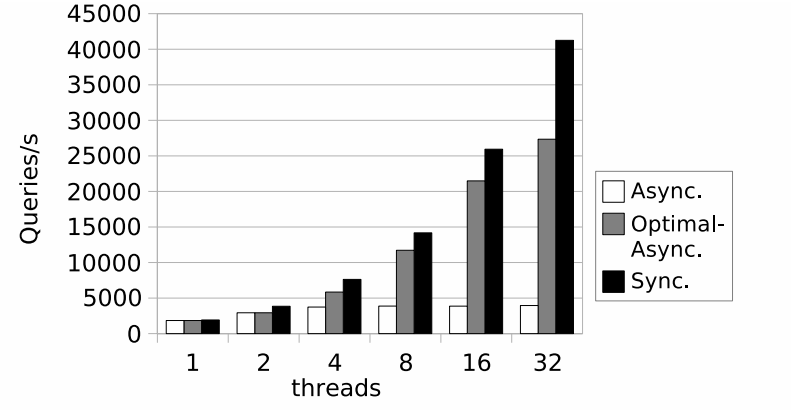


Fig. 4. Query throughput achieved by the different operation modes under study

Figure 4 highlights that even in this case, an asynchronous engine will perform much worse than the synchronous counterpart. This figure shows the query throughput achieved by the different modes. Despite the Optimal-Async model exploits both sources of parallelism – inter and intra-query parallelism – in an idealist and optimal way (thread management has been oversimplify in these simulations). Furthermore, for a given number of threads, we report the throughput achieved by the optimal combination of inter and intra-parallelism, it does not outperform the efficient synchronous model. Essentially, the overheads caused by the asynchronous thread management and the limited intra-query parallelism introduce an upper bound to the asynchronous scalability.

5.3 Fixed-Point Arithmetic: Impact on Performance and Validation

Figure 5 analyzes the impact of fixed-point arithmetic on scalability. It shows the number of queries per second that we are able to solve in the synchronous version when performing the ranking process with either floating-point or fixed point arithmetic – under high query traffic the asynchronous version behaves similarly –. As expected, the floating-point version does not scale beyond a modest number of threads.

To the best of the authors' knowledge, this is the first study that explores fixed-point arithmetic for ranking purposes. A final question here to conclude our discussion is whether the use of fixed-point operations (instead of floating-point ones) produces some effect in (1) the final set of documents selected as the answer to each query and (2) their relative position within the top K results. We evaluated this experimentally by running both fixed and floating-point document ranking functions under the same inverted file and set of queries.

We performed two tests on the set of documents generated in both cases for each query – tests on sets A for fixed-point results and B for floating-point results. The first test calculates the ratio $|A \cap B|/|B|$ for which we obtained results very close to 1; we observed average values between 0.99 and 1.0 for different and very large sets of queries. This indicates that both sets are practically identical.

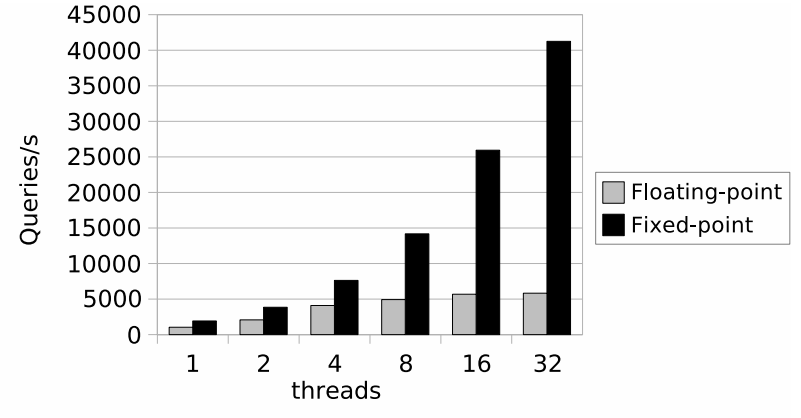


Fig. 5. Query throughput for the different numerical format representation: floating-point (gray column) and fixed-point (black column)

The second test calculates the Pearson's correlation of the sets A and B to measure the relative position of the documents in A with respect to their position in the top K query results present in B. Again we obtained values very close to 1 indicating that there are almost no difference in the relative position of the documents A in the top K results for the queries.

6 Conclusions

A logical view of the T1 processor suggests the application of the general principles of data partitioning to get the multithreaded versions of our Web Search Engine. Essentially, this partitioning is performed running queries into separate threads.

This strategy can be easily expressed with *OpenMP* directives. However, the similarities amongst the different threads may cause contention for shared resources, especially cache space and memory bandwidth. We have tried to minimize these effects with a data distribution and thread assignment strategy that looks for batching on the same processor and tries group together queries containing common terms. This strategy aims at taking advantage of the temporal data locality, and to avoid the costly memory access.

As further research we plan to increase locality by devising strategies that re-organize the way in which the chunks of size K of posting are stored in main and secondary memory in order to exploit locality. This should be made by taking into consideration how frequently the terms appears in queries together, which can be obtained from the logs that search engines maintain at their data centers.

References

1. Arusu, A., Cho, J., Garcia-Molina, H., Paepcke, A., Raghavan, S.: Searching the web. ACM Trans. 1(1), 2–43 (2001)
2. Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams, S.W., Yelick, K.A.: The landscape of parallel computing research: A view from berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley (December 2006)
3. Badue, C., Baeza-Yates, R., Ribeiro, B., Ziviani, N.: Distributed query processing using partitioned inverted files. In: Eighth Symposium on String Processing and Information Retrieval (SPIRE 2001), pp. 10–20 (2001)
4. Barroso, A., Dean, J., Olzle, U.H.: Web search for a planet: The google cluster architecture. IEEE Micro 23(2), 22–28 (2002)
5. Hennessy, J.L., Patterson, D.A.: Computer Architecture, A Quantitative Approach, 4th edn. Morgan Kaufmann Publishers Inc., San Francisco (2006)
6. Kongetira, P., Aingaran, K., Olukotun, K.: Niagara: A 32-way multithreaded sparc processor. IEEE Micro 25(2), 21–29 (2005)
7. Marin, M., Bonacic, C., Gil-Costa, V., Gomez, C.: A search engine accepting on-line updates. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 348–357. Springer, Heidelberg (2007)
8. Marin, M., Gil-Costa, V.: High-performance distributed inverted files. In: CIKM 2007: Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management, pp. 935–938. ACM, New York (2007)
9. Marin, M., Gil-Costa, V. (Sync|Async)$^+$ MPI Search Engines. In: Cappello, F., Herault, T., Dongarra, J. (eds.) PVM/MPI 2007. LNCS, vol. 4757, pp. 117–124. Springer, Heidelberg (2007)
10. Moffat, W., Webber, J., Zobel, J., Baeza-Yates, R.: A pipelined architecture for distributed text query evaluation. Information Retrieval, published on-line, 5, October (2006)
11. Olukotun, K., Hammond, L., Laudon, J.: Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency. In: Synthesis Lectures on Computer Architecture, vol. 3. Morgan and Claypool Publishers, San Francisco (2007)
12. Persin, M., Zobel, J., Sacks-Davis, R.: Filtered document retrieval with frequency-sorted indexes. Journal of the American Society for Information Science 47(10), 749–764 (1996)
13. Sheahan, D.: Developing and tuning applications on ultrasparc t1 chip multithreading systems. Technical report, Sun Microsystems. Sun BluePrints Online (October 2007)
14. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Computing Surveys 38(2) (2006)

Accomplishments and Challenges in Code Development for Parallel and Multimechanics Simulations

Tony Degroot, Robert Ferencz, Mark Havstad, Neil Hodge, Jerry Lin, Dennis Parsons, Michael Puso, Jerome Solberg, and Edward Zywicz

Lawrence Livermore National Laboratory
8000 East Ave. Livermore CA 94550

Abstract. The Methods Development Group at Lawrence Livermore National Laboratory has historically developed and supported software for engineering simulations, with a focus on nonlinear structural mechanics and heat transfer. The quality, quantity and complexity of engineering analyses have continued to increase over time as advances in chip speed and multiprocessing computers have empowered this simulation software. As such, the evolution of simulation software has seen a greater focus on multimechanics and the incorporation of more sophisticated algorithms to improve accuracy, robustness and usability. This paper will give an overview of the latest code technologies developed by the Methods Development group in the areas of large deformation transient analysis and implicit coupled codes. Applications were run on the state of the art hardware available at the national laboratories.

1 Introduction

The Methods Development Group (MDG) supports a group of roughly seventy two engineering analysts at Lawrence Livermore National Laboratory (LLNL). It supports analysts at Los Alamos National Laboratory (LANL) and limited DoD sites. LLNL is the home of some of the fastest supercomputers in the world including the world's fastest: Blue Gene/L. Much of the hardware and software development at the LLNL has been driven by the Advanced Simulation and Computing Program (ASC). ASC was created to help maintain the United States nuclear arsenal after the 1992 moratorium on nuclear testing. The MDG group has traditionally supported roughly seventy-five analysts around the laboratory in the areas of weapons thermo-structural analysis, lasers and various physics groups. The group's flagship codes: DYNA3D (explicit structural mechanics), NIKE3D (implicit structural mechanics) and the TOPAZ3D (thermal mechanics) were originally developed in the 1970's. The latest parallel codes: PARADYN and DIABLO are the parallel explicit and implicit versions of the original codes. DIABLO, is the newest code and features support for coupled structural, thermal, diffusion and electromagnetic analyses. This paper presents the latest code technologies incorporated into these codes.

The paper is organized as follows. First the issues and technologies most related to explicit/transient dynamics and the PARADYN code will be presented including: automatic/dynamic contact, extreme material deformations and coupled finite element/ meshless methods. Second, the underlying technologies and features unique to the

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 214–227, 2008.

implicit statics/dynamics multimechanics code DIABLO will be presented. The treatment of contact is still one of the group's biggest challenges, thus much of the focus of this paper. Additional aspects are presented in the areas of parallelization, solution schemes and adaptive mesh refinement. Numerous examples illustrating the latest features and run on Lab's fastest hardware will be presented.

2 Explicit Finite Elements: PARADYN

The main focus of this code is in the area of transient structural mechanics with limited thermal and fluid mechanics coupling. Many of the parallel methods underlying PARADYN are well document and very good scaling has been observed on very massively parallel runs (Fig. 1). The primary applications include the simulation of container drop tests (Fig. 2), pressure vessels (Fig. 3), infrastructure failure, (Fig. 4) automobile crash (Fig. 5) and penetration (Fig. 6). One of the main challenges in most of these problems typically is how the contact (interpenetration) constraints are handled. Issues regarding explicit contact and searching are presented here. Furthermore, because meshless methods accommodate evolving connectivity, the same dynamic partitioning can be applied to the meshless implementation.

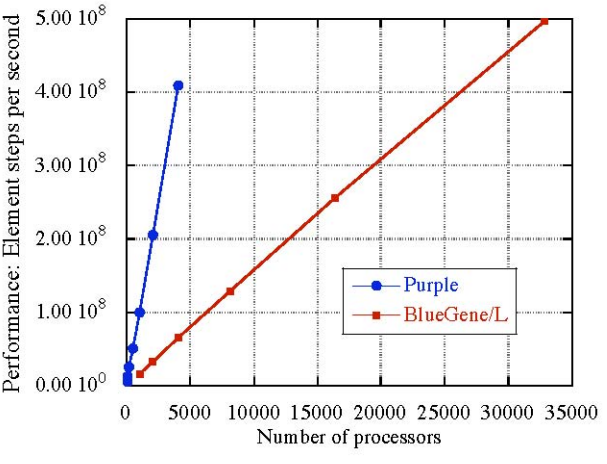


Fig. 1. Performance in element-steps per second versus number of processors of a simple 90 million element simulation using PARADYN on the ASC Purple and Blue Gene/ L platforms

2.1 Explicit Node-on-Surface Contact Formulation

The explicitly time integrated equations of motion specific to node-on-surface contact are given in Eq. 1.

$$Ma_{n+1} + f_n^{\text{int}}(x_n) + f^c = f_n^{ext} \tag{1}$$

M – diagonal mass matrix
f -force vector (internal, contact and external)
x_n, v_n, a_n nodal position, velocity and acceleration vectors at time t_n

The internal forces are the forces elements apply to nodes, the external forces are the applied nodal loads and the contact forces enforce the interpenetration constraints. Here, all time t_n quantities are known and the acceleration at time t_{n+1} is sought. The node-on-surface method computes a traction λ that forces penetrating nodes onto nearby facets. For example, in Fig. 5, the slave node S1 is forced onto segment M1-M2 and the contact force f^c is computed from the contact pressure λ at S1 and its distribution to node M1 and M2 based on where the closest point projection is. The single pass version of the method only forces slave nodes S onto opposing master segments. The symmetric or double pass version also forces master nodes M onto opposing slave segments. The contact pressure can be computed from the contact gap at time t_n using a penalty method i.e. $\lambda = \kappa g_n$. Since this gap is based on known time t_n, the contact force f^c is known and the acceleration is found easily since the mass matrix is diagonal

$$a_{n+1} = M^{-1}(f_n^{ext} + f_n^{\text{int}}(x_n) + f^c) \tag{2}$$

The velocities and positions are updated

$$v_{n+1/2} = v_{n-1/2} + \Delta t_n a_{n+1} \quad x_{n+1} = x_n + \Delta t_n v_{n+1/2} \tag{3}$$

The penalty method is often not sufficient to eliminate penetration since very high penalty values κ lower the stable time step. Here, Lagrange multipliers are used since they eliminate penetration, but don't affect the time step. The predictor-corrector strategy [1] used here computes a predictor such that no contact is active $f^c = 0$ in Eq. 1. This step yields a predicted configuration x'_{n+1} with gaps g'_{n+1}. In the corrector step, the contact tractions are treated as unknowns such that $f^c = G\,\lambda$ in Eq. 1 and the corrected displacement is computed due to these unknown contact forces

$$x_{n+1} = x'_{n+1} - \Delta t_n^2 M^{-1} G_{n+1} \lambda_{n+1} \tag{4}$$

Here the contact matrix *G* is based on segment normal vectors (Fig. 5) and is also used to compute the nodal gap vector i.e. $g = Gx$. Multiplying Eq. 4 by G yields the nodal gap vector based on the predicted configuration and unknown contact pressure

$$g_{n+1} = g'_{n+1} - \Delta t^2 G_{n+1}{}^t \; M^{-1} G_{n+1} \; \lambda_{n+1} \tag{5}$$

The nodal gaps must satisfy the Kuhn Tucker conditions:

$$\lambda_{n+1} \geq 0, \quad g_{n+1} \geq 0, \quad g_{n+1}\lambda_{n+1} \geq 0 \tag{6}$$

i.e. contact pressure must be compressive, gaps must be open and only closed gaps have contact pressure. This is a mathematical programming problem and is solved via a parallel, constrained, preconditioned conjugate gradient method.

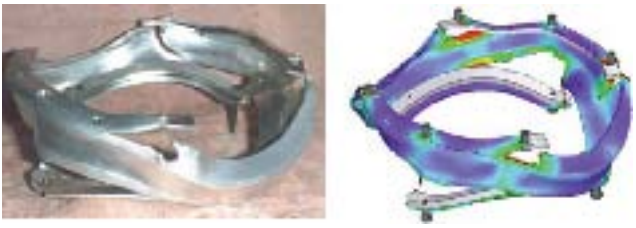


Fig. 2. Transportation container flange detail (Courtesy of Dan Badders, LLNL)

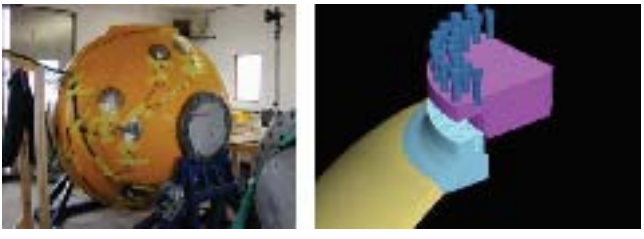


Fig. 3. Hydrodynamic containment vessel (LANL Weapons Engineering)

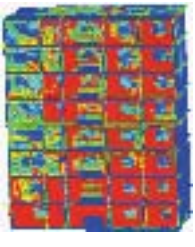


Fig. 4. Blast loading on apartment building, 30 millioin degrees of freedom (P. Papados, U.S. Army ERDC)

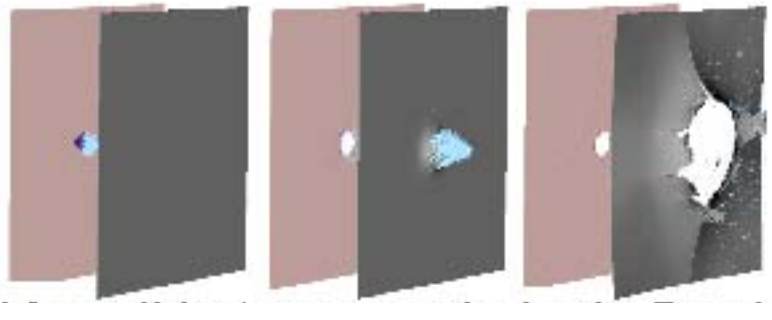


Fig. 5. Sequence of deformation as penetrator goes through two plates. The secondary damage is a result of the inertia from the fragments from the first penetration. That is, new contact surface (fragments) and particles from the first penetration will impact the second plate.

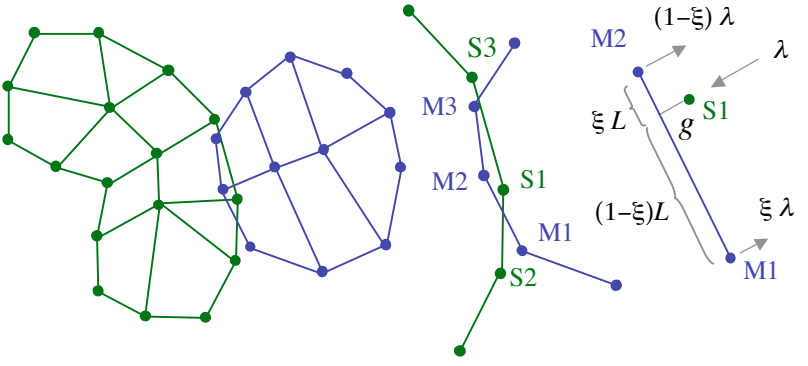


Fig. 6. (a) Impacting bodies. (b) Intersecting slave and master contact surfaces. (c) Contact gap based on closest point, contact forces at slave node S1 and master nodes M1 and M2.

2.2 Parallel Contact Search Algorithm

The node-on-segment contact methods described in the previous section require a search for the "nearest" master segments for each slave node (and slave segment for each master node for the double pass version). This process can be one of the most time consuming in an explicit finite element code. Master segments are generated automatically by computing the free/exposed element faces on the mesh. For large deformation problems, elements will become damaged and no longer active. This "element erosion" defines new master segments that must be accounted for. The parallel contact algorithm is based on a static domain decomposition of the mesh and a dynamic decomposition of the contact segments as follows:

1. Compute static decomposition of the mesh using METIS [2]. This is done once at problem initialization. Nodes on partition boundaries are assigned a home processor and are considered shared on remaining partitions.
2. On each static partition, define all free facets to be candidate contact segments and define all nodes attached to these facets to be candidate slave nodes. This is a double pass algorithm.
3. Based on a characteristic distance, subdivide the entire domain into bins and loop over all slave nodes and master segments to determine which bin it resides in.
4. A graph structure where each bin is a vertex and each master segment that overlaps bins is considered a connective edge,
5. Use METIS [2] to partition this graph structure to minimize edge cuts and assign bins to a partition.
6. Assign each "dynamic" partition to the processor that most of its contact nodes have for the "static" partition.
7. A serial algorithm (e.g. bucket sort) is done in each dynamic partition to find contact node-segment pairs. Segments that overlap partitions are shared, hence the METIS [2] partitioning reduces the search size.
8. Contact forces are computed on each partition and then communicated to the home processor. The home processor then communicates the contact forces to the remaining shared partitions (processors) if any.

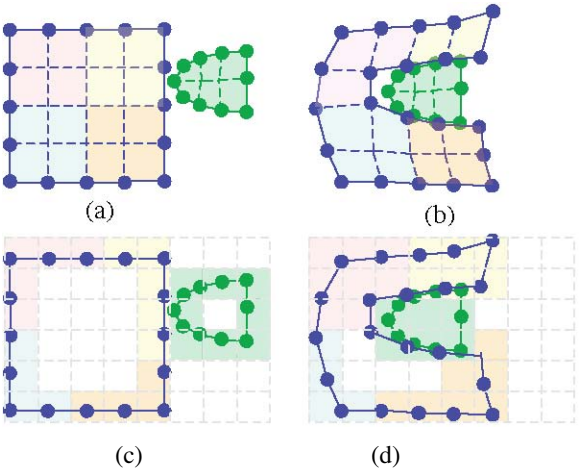


Fig. 7. (a) Static decomposition of mesh in un-deformed configuration. (b) Large deformations cause element erosion due to damage i.e. elements are eliminated. (c) The candidate contact segments are binned, partitioned according to METIS and placed on processors based on the home processors of the nodes in each bin. (d) This process is done dynamically as elementerosion defines new candidate master segments and large motions change the graph structure ofthe contact bins.

This process is illustrated in Fig. 7. The simulation in Fig. 5 used this parallel search algorithm with the element erosion and the Lagrange multiplier contact. In that, the elements fail but the surrounding nodes are still considered point masses so that momentum is conserved. These point masses are included as potential contact node sin the process described above and cause secondary damage in the following plate.

2.3 Meshless Methods

The meshless methods were developed to handle very large deformation problems where mesh tangling becomes an issue. That is, when elements get so distorted due to deformation, they are no longer usable (e.g. the invert). Meshless methods do not rely on elements to parameterize deformation but instead use shape functions that can operate on arbitrary point clouds. The overlap (Fig. 8) of elliptical meshless shape functions defines the graph structure of the discretization [3]. Since the meshless particles flow such that new support overlaps need to be defined on a regular basis for

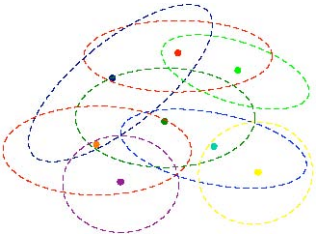


Fig. 8. Overlapping elliptical supports of meshless method

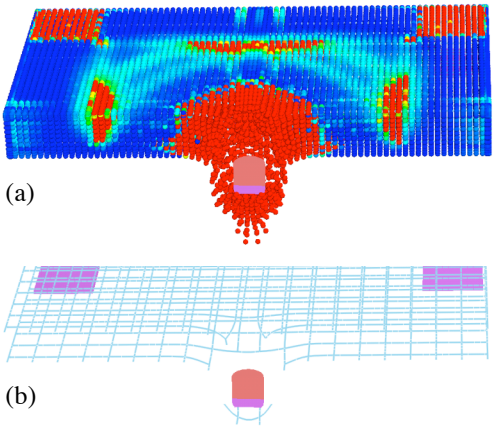


Fig. 9. (a) Simulation of penetration of a reinforced concrete (tensile damage shown in red) (b) Rebar (shown alone) was attached to particles in the simulation and eventually fails upon exit

large deformation problems, a dynamic partitioning method identical to that for contact should be used. Fig. 9 shows a simulation of a steel penetrator going through a concrete slab with rebar.

3 Implicit Finite Elements: DIABLO

Different strategies exist for how mechanics coupling is implemented in simulations codes. One of the simplest strategies is to take independent codes and then pass a limited amount of data (e.g. node positions, node temperatures and time) through an interface. This is the strategy for the coupling the DYNA3D structural mechanics mechanics code with the TOPAZ thermal mechanics code and the finite volume fluids code GEMINI. The codes themselves are standalone with minimum sharing of data structures and code reuse. This model can get more complicated when adaptive meshing (AMR) is employed since some common definition for the octree mesh needs to be defined. Because many of the operations needed to do error estimating and data remapping need to be written for data manipulation, a model that employs an ample amount of code reuse between the different mechanics becomes more tractable. As such, a single data structure for all the different mechanics would make this code re-use more practical. DIABLO is the latest MDG code project and employs the latter strategy. DIABLO currently has coupled solid mechanics, thermal mechanics, electromagnetics and diffusion along with adaptive meshing. Fig. 6 shows an example where AMR is applied to a coupled thermal structural simulation. Example applications include metal forming, rail gun etc. As with PARADYN, the contact algorithms are considered a key component of the code development of DIABLO and are highlighted here.

3.1 Implict Finite Elements: Segment-to-Segment Contact

Node-on-segment algorithms are simple but have a number of flaws. In particular, they don't transmit stresses or fluxes smoothly across the boundary. This is particularly important for solid mechanics where non-smooth force transmission hinders converge of the implicit non-linear algorithm and in electromagnetics where Nedelec edge elements are used. Referring to Fig. 11, the mortar segment-to-segment algorithm for solid mechanics [4,5] computes a nodal gap based on the integral

$$g_A = \int N_A (x^s - x^m)\, d\Gamma \tag{7}$$

Here the contact traction can be computed via a penalty $\lambda_A = \kappa g_A$ or an augmented Lagrangian $\lambda_A^{i+1} = \kappa g_A^i + \lambda_A^i$ computed through an Uzawa algorithm. Now the static, implicit discrete equations of motion are written

$$f_{n+1}^{\text{int}}(x_{n+1}) + f_{n+1}^{c} - f_{n+1}^{ext} = 0 \tag{8}$$

where the unknown configuration x_{n+1} is typically solved for using the linearized form of Eq. 8 in a Newton Raphson (or Quasi-Newton) scheme. The additional mechanics types compute an analogous discrete balance equation. Fig. 12 illustrates the nonlinear algorithm used in DIABLO. Fig. 13 demonstrates the robustness of mortar segment-to-segment contact method.

3.2 Parallel Contact Search Algorithm

The parallel implementation of this contact was kept simple for the sake of development and because communication costs for an implicit code are small compared to the solution of simultaneous equations. Here we choose to build an entirely static decomposition using METIS [2] and then add shared nodes from relevant contact surfaces on each partition (Fig. 14). Now a serial algorithm (e.g. bucket sort) can be performed on each processor for the contact search. Contact forces are only computed on contact home nodes and then scattered via point-to-point communications to shared nodes.

3.3 Multimechanics Examples

One of the main applications for the coupled solid electromagnetics is the rail gun. Here a high voltage is applied between two rails and an armature carries current between the rails (Fig. 15). Because of the transient time for the current to penetrate the rails and armature (magnetic diffusion), the current travels along the skin of the rails (i.e. skin effect). Ampere's law $\nabla \times \mathbf{B} = \mu \mathbf{J}$ predicts that the resulting magnet $\mathbf{B}$ field will be relatively high (pointing out) on the inside of the circuit (red) but small inside the surrounding iron armature and rails due to the low magnet permeability μ of the iron and high permeability μ of the air. This produces a Lorentz

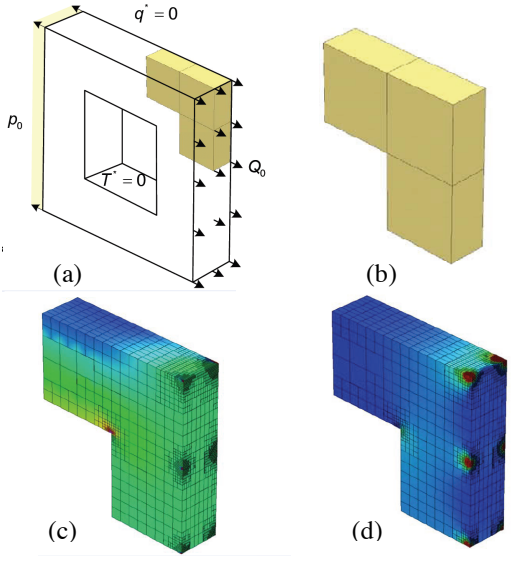


Fig. 10. Thermal-structural AMR example: (a) ring with (tensile) pressure loading (over faces) and concentrated thermal fluxes **Q**, (on nodes) on left and right side. (b) initial quarter symmetry mesh (c) effective stress and (d) temperature on final adapted mesh. Note that refinement was made near stress concentrations on the inner corner and locations where nodal flux loads reside.

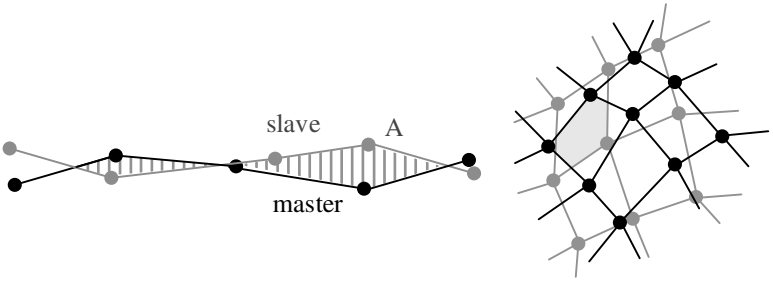


Fig. 11. Segment-to-segment contact computes the weighted volume between adjacent facets to get the nodal gap. This is simple in 2D (left). More sophisticated 3D algorithms require the intersection of adjacent slave and master segments to compute the nodal gap.

force per unit length, $\mathbf{F} = \mathbf{J} \times \mathbf{B}$ on the armature propelling it outward. Referring to Fig. 15, the model uses a grid for the armature (blue), two grids for the rails and four grids for the air. Only the rail and armature meshes consist of solid elements. All meshes are required to capture the magnetic field. Two fine (inner) air meshes move with the armature and interact with the surrounding air and rail through electromagnetic (mortar) contact surfaces. Both mechanical and electromagnetic

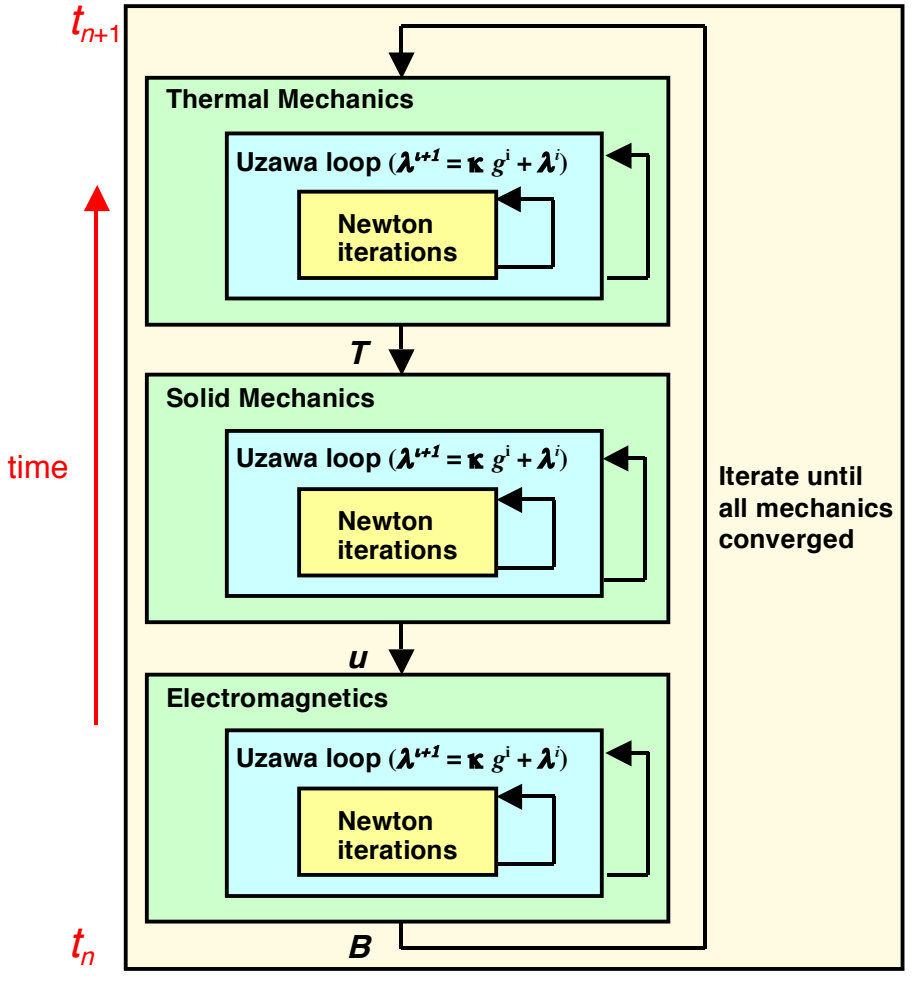


Fig. 12. Flow of nonlinear solution algorithm over a time step. A Uzawa loop is included to enforce augmented Lagrange type contact for each mechanics.

contact is used between the armature and the two rails Snapshots of the 3D simulation are shown in Fig. 16 as the armature slides across the rails. Quarter symmetry is used to reduce the model size such that only the top, left-hand side of the mesh is shown. Here it is confirmed that the **B** field is very high at the back of the armature (i.e. fine mesh in Fig. 15) and low in the surrounding air. The problem had ~4.5 million degrees of freedom and was run on 16 processors on the ASC Purple platform in 20 hours.

Another important application is in nuclear engineering (Fig. 17). Here, the fuel bundle is composed of hexagonal fuel rods (metal alloy tubes filled with nuclear fuel).The fuel is meshed as a homogenous material and acts as a neutron heat source. The rods are inserted into three restraint plates with hexagonal slots to constrain the

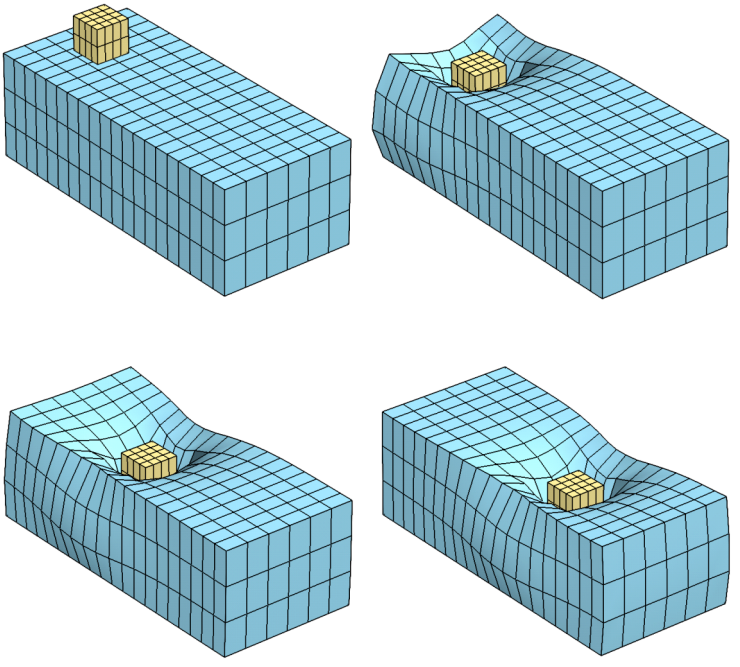


Fig. 13. Sequence of deformations of steel block penetrating rubber block and sliding. Because of the sharp corners and the "bumpy" segments, this problem can't be solved using node-on segment with an implicit scheme.

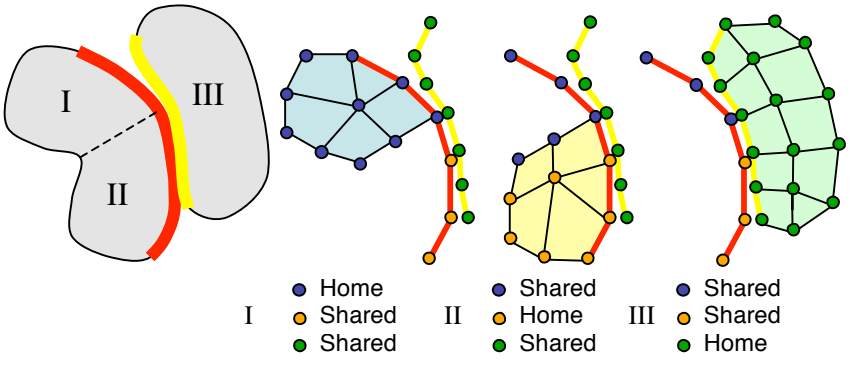


Fig. 14. DIABLO builds a static domain decompositions and then adds shared nodes to each partition so that all relevant contact nodes reside on the partition. So, for example, partition II computes contact gaps on its three (orange) home contact nodes using coordinates from its shared (green,blue) nodes. Forces are then communicated back to shared nodes.

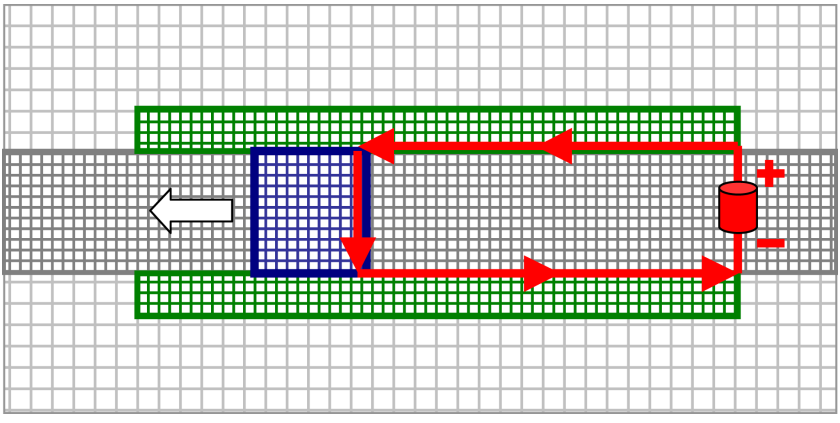


Fig. 15. A 2D slice of a 3D mesh used for electromagnetic rail gun. A large voltage is applied across the rails (green) and the resulting current (red) stays close to the surface as it passes across the armature (blue).

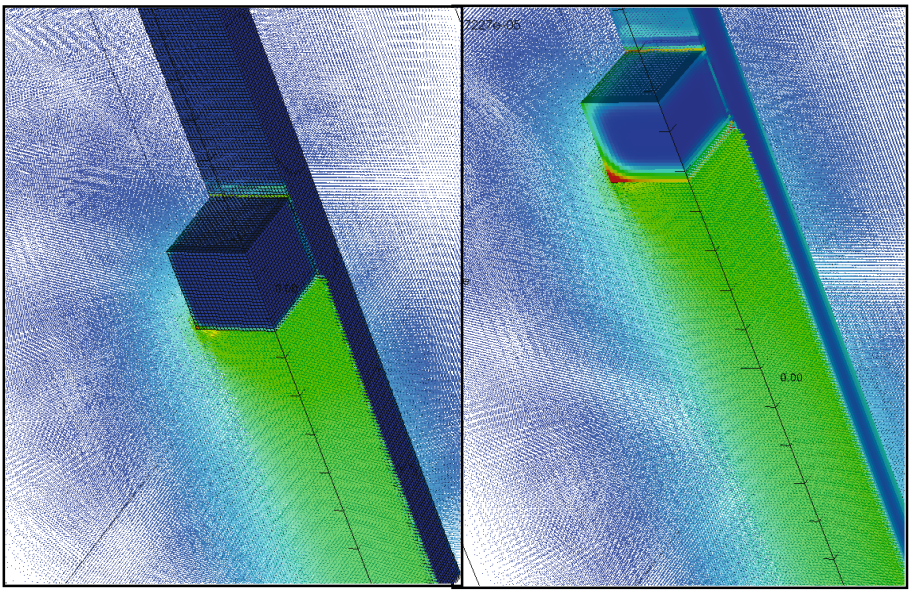


Fig. 16. A quarter symmetry simulation of the 3D rail gun simulation. The trailing **B** field is very high (green) whilst the **B** field in the air (blue) is very low. Only a little current is visible in the armature and rail at the early time, but it becomes particularly apparent in the outside portion (red) of the armature at the later time.

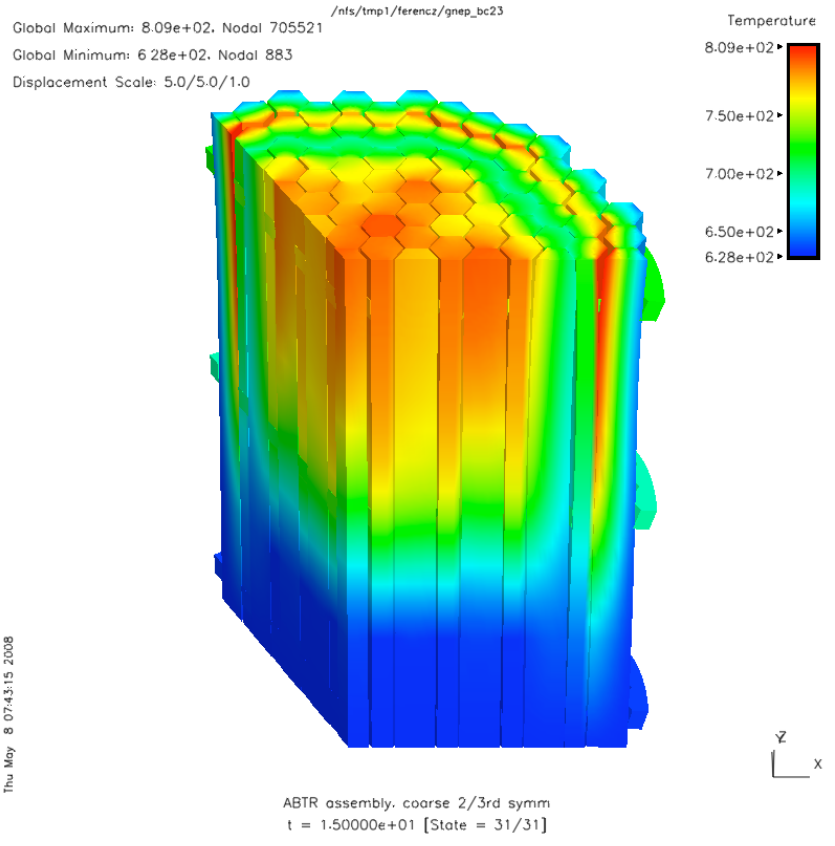


Fig. 17. Nuclear fuel bundle (one-third symmetry). Neutron heat sources cause thermal gradient and bending of fuel rods forming large gaps in contact surfaces between adjacent rods.

individual rods. The model in Fig. 17 is one-third symmetry and includes 390 contact surfaces between each of the 75 rods and also the slots in the restraint plates. The boundary conditions include neutron heat sources near the top center of the fuel bundle and thermal boundary conditions on the outer edges of the constraint plates. A separate neutronics package determined the heat sources in the model. The model was over 2 million degrees of freedom and was solved using 32 partitions on 32 (8 processor) nodes on the ASC Purple machine. Each partition had a dedicated processor and four threads per node were used for the parallel linear direct solver. The thermal gradient that results from the heating causes bending in the individual rods such that many large gaps open up between the adjacent rods near the top.

4 Discussion

Different methods were presented for doing parallel contact with applications in both large deformations with failure and multimechanics. A novel dynamic partitioning

algorithm was presented for the contact searching and meshless particle methods. An overview of a one-of-a-kind parallel implicit solid-thermal-electromagnetics code was presented along with its novel contact algorithms capabilities. Future work includes embedded mesh techniques and coupled mechanics on different meshes.

Acknowledgment

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

References

1. Zywicz, E., Puso, M.A.: A General Predictor-Corrector Solver for Explicit Finite-Element Contact. Int. J. Numer. Meth. Eng. 44, 439–459 (1999)
2. Karypis, G., Kumar, V.: Metis: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices, version 4.0, University of Minnesota, Department of Computer Science (1998)
3. Puso, M.A., Chen, J.S., Zywicz, E., Elmer, W.: Meshfree and Finite Element Nodal Integration Methods. Int. J. Numer. Meth. Eng. 74, 416–446 (2008)
4. Puso, M.A., Laursen, T.A.: A Mortar Segment-to-Segment Contact Method for Large Deformation Solid Mechanics. Comput. Meth. Appl. M. 193, 601–629 (2004)
5. Puso, M.A., Laursen, T.A.: A Mortar Segment-to-Segment Frictional Contact Method for Large Deformations. Comput. Meth. Appl. M. 193, 4891–4913 (2004)

An Algorithm-by-Blocks for SuperMatrix Band Cholesky Factorization

Gregorio Quintana-Ortí[1], Enrique S. Quintana-Ortí[1], Alfredo Remón[1], and Robert A. van de Geijn[2]

[1] Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071Castellón, Spain
{gquintan,quintana,remon}@icc.uji.es
[2] Department of Computer Sciences, The University of Texas at Austin, Austin, Texas 78712
rvdg@cs.utexas.edu

Abstract. We pursue the scalable parallel implementation of the factorization of band matrices with medium to large bandwidth targeting SMP and multi-core architectures. Our approach decomposes the computation into a large number of fine-grained operations exposing a higher degree of parallelism. The SuperMatrix run-time system allows an out-of-order scheduling of operations that is transparent to the programmer. Experimental results for the Cholesky factorization of band matrices on two parallel platforms with sixteen processors demonstrate the scalability of the solution.

Keywords: Cholesky factorization, band matrices, high-performance, dy- namic scheduling, out-of-order execution, linear algebra libraries.

1 Introduction

How to extract parallelism from linear algebra libraries is being reevaluated with the emergence of SMP architectures with many processors, multi-core systems that will soon have many cores, and hardware accelerators such as the Cell BE processor or graphics processors (GPUs). In this note, we demonstrate how techniques that have shown to be extremely useful for the parallelization of dense factorizations in this context [7,8,17,18,6,5] can also be extended for the factorization of band matrices [19]. The result is an algorithm-by-blocks that yields high performance and scalability for matrices of moderate to large bandwidth while keeping the implementation simple by various programmability measures. To illustrate our case, we employ the Cholesky factorization of band symmetric positive definite matrices as a prototypical example. However, the same ideas apply to algorithms for the LU and QR factorization of band matrices.

The contributions of this paper include the following:

- We demonstrate that high performance can be attained by programs coded at a high level of abstraction, even by algorithms for complex operations like the factorization of band matrices and on sophisticated environments like many-threaded architectures.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 228–239, 2008.

- We show how the SuperMatrix run-time system supports out-of-order computation on blocks transparent to the programmer leading to a solution which exhibits superior scalability for band matrices.
- We also show how the FLASH extension of FLAME supports storage by blocks for band matrices different from the commonly-used packed storage used in LAPACK [1].
- We compare and contrast a traditional blocked algorithm for the band Cholesky factorization to a new algorithm-by-blocks.

This paper is structured as follows. In Section 2 we describe a blocked algorithm for the Cholesky factorization of a band matrix which reflects the state-of-the-art for this operation. Then, in Section 3, we present an algorithm-by-blocks which advances operations that are in the critical path from "future" iterations. The FLAME tools employed to implement this algorithm are outlined in Section 4. In Section 5 we demonstrate the scalability of this solution on a CC-NUMA with sixteen Intel Itanium2 processors and an SMP with 8 AMD Opteron (dual core) processors. Finally, in Section 6 we provide a few concluding remarks.

In the paper, matrices, vectors, and scalars are denoted by upper-case, lower-case, and lower-case Greek letters, respectively. Algorithms are given in a notation that we have developed as part of the FLAME project [3]. If one keeps in mind that the thick lines in the partitioned matrices and vectors relate to how far the computation has proceeded, we believe the notation is mostly intuitive. Otherwise, we suggest that the reader consult some of these related papers.

2 Computing the Cholesky Factorization of a Band Matrix

Given a symmetric positive definite matrix A of dimension $n \times n$, its Cholesky factorization is given by $A = LL^T$, where L is an $n \times n$ lower triangular matrix. (Alternatively, A can be decomposed into the product $A = U^T U$ with U an $n \times n$ upper triangular matrix, a case that we do not pursue further.) In case A presents a band structure with upper and lower bandwidth k_d (that is, all entries below the $k_d + 1$ subdiagonal and above the $k_d + 1$ superdiagonal are zero), then L presents the same lower bandwidth as A. Exploiting the band structure of A when $k_d \ll n$ leads to important savings in both storage and computation. This was already recognized in LINPACK and later in LAPACK which includes unblocked and blocked routines for the Cholesky factorization of a band matrix.

2.1 The Blocked Algorithm in Routine PBTRF

It is well-known that high performance can be achieved in a portable fashion by casting algorithms in terms of matrix-matrix multiplication [1,11]. Figure 1 illustrates how the LAPACK blocked routine PBTRF does so for the Cholesky factorization of a band matrix with non-negligible bandwidth. For simplicity we

Algorithm: $[A] := \text{band\_Cholesky}_{\text{BLK}}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c|c} A_{TL} & \star & \\ \hline A_{ML} & A_{MM} & \star \\ \hline & A_{BM} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0 and A_{MM} is $k_d \times k_d$

while $m(A_{TL}) < m(A)$ **do**

Determine block size n_b

Repartition

$$\left(\begin{array}{c|c|c} A_{TL} & \star & \\ \hline A_{ML} & A_{MM} & \star \\ \hline & A_{BM} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c|c|c} A_{00} & \star & \star & & \\ \hline A_{10} & A_{11} & \star & \star & \\ \hline A_{20} & A_{21} & A_{22} & \star & \star \\ \hline & A_{31} & A_{32} & A_{33} & \star \\ \hline & & A_{42} & A_{43} & A_{44} \end{array}\right)$$

where A_{11}, A_{33} are $n_b \times n_b$, and A_{22} is $k \times k$, with $k = k_d - n_b$

$A_{11} = L_{11}L_{11}^T$	Dense Cholesky factorization
$A_{21} := A_{21}L_{11}^{-T} (= L_{21})$	Triangular system solve
$A_{31} := A_{31}L_{11}^{-T} (= L_{31})$	Triangular system solve with triangular solution
$A_{22} := A_{22} - L_{21}L_{21}^T$	Symmetric rank-k update
$A_{32} := A_{32} - L_{31}L_{21}^T$	Triangular matrix-matrix product
$A_{33} := A_{33} - L_{31}L_{31}^T$	Symmetric rank-n_b update

Continue with

$$\left(\begin{array}{c|c|c} A_{TL} & \star & \\ \hline A_{ML} & A_{MM} & \star \\ \hline & A_{BM} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c|c|c} A_{00} & \star & \star & & \\ \hline A_{10} & A_{11} & \star & \star & \\ \hline A_{20} & A_{21} & A_{22} & \star & \star \\ \hline & A_{31} & A_{32} & A_{33} & \star \\ \hline & & A_{42} & A_{43} & A_{44} \end{array}\right)$$

endwhile

Fig. 1. Blocked algorithm for the Cholesky factorization of a band matrix

consider there and hereafter that n and k_d are exact multiples of k_d and n_b, respectively. Provided $n_b \ll k_d$ most of the computations in the algorithm are cast into the symmetric rank-k update of A_{22}.

Upon completion of the factorization using the algorithm in the figure, the elements of L overwrite the corresponding entries of A. The "$\star$" symbols in the figure denote symmetric parts in the upper triangle of A which are not accessed/referenced.

2.2 Packed Storage in Routine PBTRF

Routine PBTRF employs a packed format to save storage. Specifically, the symmetry of A requires only its lower (or upper) triangular part to be stored, which is saved following the pattern illustrated in Figure 2 (right). As they are

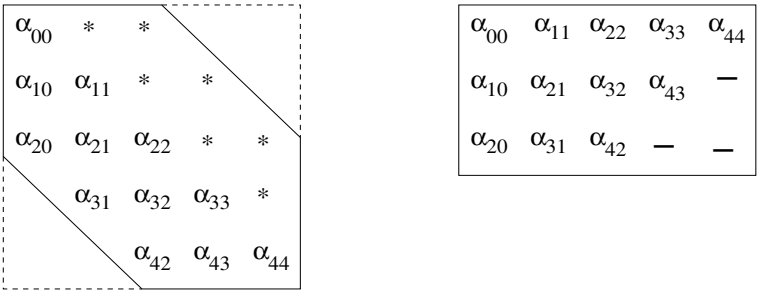


Fig. 2. Symmetric 5×5 band matrix with bandwidth $k_d = 2$ (left) and packed storage used in LAPACK (right). The '$*$' symbols denote the symmetric entries of the matrix which are not stored in the packed format.

computed, the elements of L overwrite the corresponding entries of A in the packed matrix.

Due to A_{31}/L_{31} having only their upper triangular parts stored, some operations in the actual implementation of the algorithm in Figure 1 need special care, as described next. In particular, in order to solve the triangular linear system for A_{31}, a copy of the upper triangular part of A_{31} is first obtained in an auxiliary workspace W of dimension $n_b \times n_b$ with its subdiagonal entries set to zero; the BLAS-3 solver TRSM is then used to obtain $W := WL_{11}^{-T}$ $(= L_{31})$. Next, the update of A_{32} is computed as a general matrix product using BLAS-3 kernel GEMM to yield $A_{32} := A_{32} - WL_{21}^T$. Finally, the update of A_{33} is obtained using BLAS-3 kernel SYRK as $A_{33} := A_{33} - WW^T$, and the upper triangular part of W is written back to A_{31}.

2.3 Parallelism within the BLAS

Blocked implementations of the band Cholesky factorization are typically written so that the bulk of the computation is performed by calls to the *Basic Linear Algebra Subprograms* (BLAS), a standardized interface to routines that carry out operations as matrix-vector (level-2 BLAS) and matrix-matrix multiplication (level-3 BLAS).

Parallelism can be attained within each call to a BLAS routine with the following benefits:

- The approach allows legacy libraries, such as LAPACK, to be used without change.
- Parallelism within suboperations, e.g., the update of A_{11}–A_{33} in Figure 1, can be exploited through multithreaded implementations of BLAS. However, note that in practice $k_d \gg n_b$ and n_b is typically small so that the major bulk of the computations is in the update of A_{22} while the remaining operations may be too small to gain any benefit from the use of a multithreaded implementation of BLAS.

Disadvantages, on the other hand, include:

- The parallelism achieved is only as good as the underlying multithreaded implementation of the BLAS.
- The end of each call to a BLAS operation becomes a synchronization point (a barrier) for the threads. In [20] it is shown how the updates of A_{21}, A_{31} can be merged into a single triangular linear system solve and the updates A_{22}, A_{32}, and A_{33} into a single symmetric rank-k_d update, so that a coarser grain of parallelism is obtained and the number of synchronization points is diminished. The performance increase which can be gained from this approach is modest, within 5–10% depending on the bandwidth of the matrix and the architecture.
- For many operations the choice of algorithmic variant can severely impact the performance that is achieved.

In the next section we propose an algorithm composed of operations with finer granularity to overcome these difficulties.

3 An Algorithm-by-Blocks

Since the early 1990s, various researchers [10,12,13,16] have proposed that matrices should be stored by blocks as opposed to the more customary column-major storage used in Fortran and row-major storage used in C. Doing so recursively is a generalization of that idea. The original reason was that by storing matrices contiguously a performance benefit would result. More recently, we have proposed that the blocks should be viewed as units of data and operations with blocks as units of computation [7,9]. In the following we show how to decompose the updates in the algorithm for the band Cholesky factorization so that an algorithm-by-blocks results which performs all operations on "tiny" $n_b \times n_b$ blocks.

For our discussion below, assume $k = pn_b$ for the blocked algorithm in Figure 1. Then, given the dimensions imposed by the partitionings on A,

$$\left(\begin{array}{c|c|c} A_{11} & \star & \star \\ \hline A_{21} & A_{22} & \star \\ \hline A_{31} & A_{32} & A_{33} \end{array}\right) \rightarrow \left(\begin{array}{c|ccccc} A_{11} & \star & \star & \star & \star & \star \\ \hline A_{21}^{0} & A_{22}^{00} & \star & \star & \star & \star \\ A_{21}^{1} & A_{22}^{10} & A_{22}^{11} & \star & \star & \star \\ \vdots & \vdots & \vdots & \ddots & & \\ A_{21}^{p-1} & A_{22}^{p-1,0} & A_{22}^{p-1,1} & \cdots & A_{22}^{p-1,p-1} & \star \\ \hline A_{31} & A_{32}^{0} & A_{32}^{1} & \cdots & A_{31}^{p-1} & A_{33} \end{array}\right),$$

where all blocks are $n_b \times n_b$. Therefore, the update $A_{21} := A_{21} L_{11}^{-T}$ in the blocked algorithm can be decomposed into

$$\left(\begin{array}{c} A_{21}^{0} \\ A_{21}^{1} \\ \vdots \\ A_{21}^{p-1} \end{array}\right) := \left(\begin{array}{c} A_{21}^{0} \\ A_{21}^{1} \\ \vdots \\ A_{21}^{p-1} \end{array}\right) L_{11}^{-T}, \tag{1}$$

which corresponds to p triangular linear systems on $n_b \times n_b$ blocks. Similarly, the update $A_{22} := A_{22} - L_{21}L_{21}^T$ becomes

$$\begin{pmatrix} A_{22}^{00} & \star & \star & \star \\ A_{22}^{10} & A_{22}^{11} & \star & \star \\ \vdots & \vdots & \ddots & \star \\ A_{22}^{p-1,0} & A_{22}^{p-1,1} & \dots & A_{22}^{p-1,p-1} \end{pmatrix} := \begin{pmatrix} A_{22}^{00} & \star & \star & \star \\ A_{22}^{10} & A_{22}^{11} & \star & \star \\ \vdots & \vdots & \ddots & \star \\ A_{22}^{p-1,0} & A_{22}^{p-1,1} & \dots & A_{22}^{p-1,p-1} \end{pmatrix} - \begin{pmatrix} A_{21}^0 \\ A_{21}^1 \\ \vdots \\ A_{21}^{p-1} \end{pmatrix} \begin{pmatrix} A_{21}^0 \\ A_{21}^1 \\ \vdots \\ A_{21}^{p-1} \end{pmatrix}^T, \tag{2}$$

where we can identify p symmetric rank-n_b updates (for the $n_b \times n_b$ diagonal blocks) and $(p^2/2 - p/2)$ general matrix products (for the $n_b \times n_b$ subdiagonal blocks). Finally, the update $A_{32} := A_{32} - L_{31}L_{21}^T$ is equivalent to

$$\begin{pmatrix} A_{32}^0 & A_{32}^1 & \dots & A_{31}^{p-1} \end{pmatrix} := \begin{pmatrix} A_{32}^0 & A_{32}^1 & \dots & A_{31}^{p-1} \end{pmatrix} - A_{31} \begin{pmatrix} A_{21}^0 \\ A_{21}^1 \\ \vdots \\ A_{21}^{p-1} \end{pmatrix}^T \tag{3}$$

which, given the upper triangular structure of A_{31}, corresponds to p triangular matrix-matrix products of dimension $n_b \times n_b$.

The first key point to realize here is that all the operations on blocks in (1) are independent and therefore can be performed concurrently. The same holds for the operations in (2) and also for those in (3). By decomposing the updates of A_{21}, A_{22}, and A_{32} as in (1)–(3) more parallelism is exposed at the block level in the algorithm in Figure 1.

The second key point is that some of the block operations in (1) can proceed in parallel with block operations in (2) and (3). Thus, for example, $A_{21}^1 := A_{21}^1 L_{11}^{-1}$ is independent from $A_{22}^{00} := A_{22}^{00} - A_{21}^0 (A_{21}^0)^T$ and both can be computed in parallel. This is a fundamental difference compared with a parallelization entirely based on a parallel (multithreaded) BLAS, where each BLAS call is a synchronization point so that, e.g., no thread can be updating (a block of) A_{22} before the update of (all blocks within) A_{21} is completed.

4 The FLAME Tools

In this section we briefly review some of the tools that the FLAME project puts at our disposal.

4.1 FLAME

FLAME is a methodology for deriving and implementing dense linear algebra operations [3]. The (semiautomatic) application of this methodology produces provably correct algorithms for a wide variety of linear algebra computations.

The use of the Application Programming Interface (API) for the C programming language allows an easy translation of FLAME algorithm to C code, as illustrated for dense linear algebra operations in [4].

4.2 FLASH

One naturally thinks of matrices stored by blocks as matrices of matrices. As a result, if the API encapsulates information that describes a matrix in an object, as FLAME does, and allows an element in a matrix to itself be a matrix object, then algorithms over matrices stored by blocks can be represented in code at the same high level of abstraction. Multiple layers of this idea can be used if multiple hierarchical layers in the matrix are to be exposed. We call this extension to the FLAME API the FLASH API [15]. Examples of how simpler operations can be transformed from FLAME to FLASH implementations can be found in [7,9].

The FLASH API provides a manner to store band matrices that is conceptually different from that of LAPACK. Using the FLASH API, a blocked storage is easy to implement where only those $(n_b \times n_b)$ blocks with elements within the (nonzero) band are actually stored. The result is a packed storage which roughly requires same the order of elements as the traditional packed scheme but which decouples the logical and the physical storage patterns, yielding higher performance. Special storage schemes for triangular and symmetric matrices can still be combined for performance or to save space within the $n_b \times n_b$ blocks.

4.3 SuperMatrix

Given a FLAME algorithm implemented in code using the FLAME/C interface, the SuperMatrix run-time system first builds a Directed Acyclic Graph (DAG) that represents all operations that need to be performed together with the dependencies among these. The run-time system then uses the information in the DAG to schedule operations for execution dynamically, as dependencies are fulfilled. These two phases, construction of the DAG and scheduling of operations, can proceed completely transparent to the specific implementation of the library routine. For further details on SuperMatrix, see [7,9].

We used OpenMP to provide multithreading facilities where each thread executes asynchronously. We have also implemented SuperMatrix using the POSIX threads API to reach a broader range of platforms.

Approaches similar to SuperMatrix have been described for more general irregular problems in the frame of the Cilk project [14] (for problems that can be easily formulated as divide-and-conquer, unlike the band Cholesky factorization), and for general problems also but with the specific target of the Cell processor in the CellSs project [2].

5 Experiments

In this section, we evaluate two implementations for the Cholesky factorization of a band matrix with varying dimension and bandwidth. Details on the platforms

that were employed in the experimental evaluation are given in Table 1. Both architectures consist of a total of 16 CPUs: SET is a CC-NUMA platform with 16 processors while NEUMANN is an SMP of 8 processors with 2 cores each. The peak performance is 96 GFLOPS (96×10^9 flops per second) for SET and 70.4 GFLOPS for NEUMANN.

Table 1. Architectures (top) and software (bottom) employed in the evaluation

Platform	Architecture	Frequency (GHz)	L2 cache (KBytes)	L3 cache (MBytes)	Total RAM (GBytes)
SET	Intel Itanium2	1.5	256	4096	30
NEUMANN	AMD Opteron	2.2	1024	–	63

Platform	Compiler	Optimization flags	BLAS	Operating System
SET	`icc` 9.0	`-O3`	MKL 8.1	Linux 2.6.5-7.244-sn2
NEUMANN	`icc` 9.1	`-O3`	MKL 9.1	Linux 2.6.18-8.1.6.el5

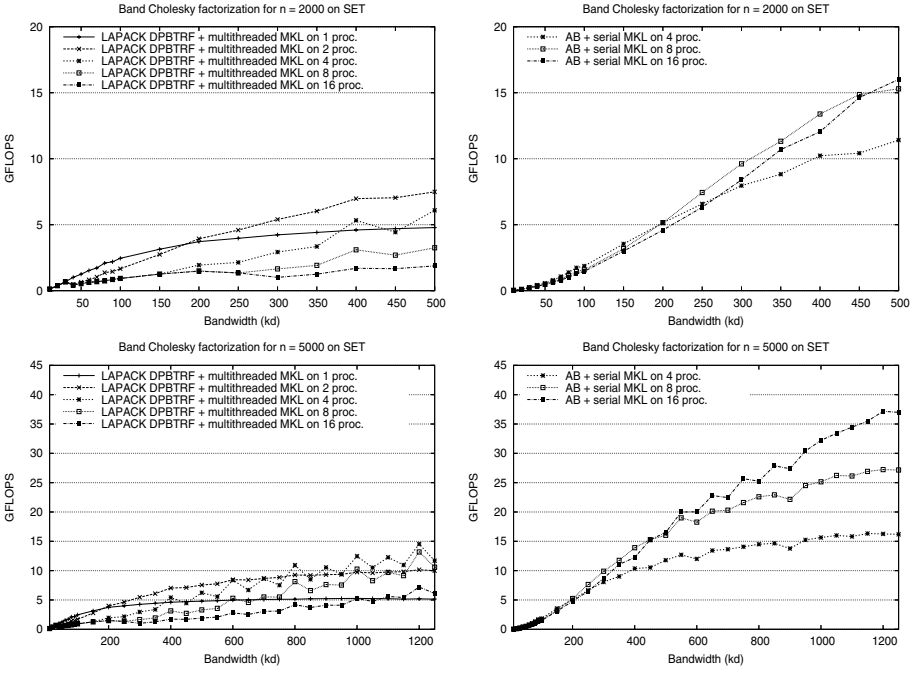


Fig. 3. Performance of the band Cholesky factorization algorithms on 1, 2, 4, 8, and 16 CPUs of SET

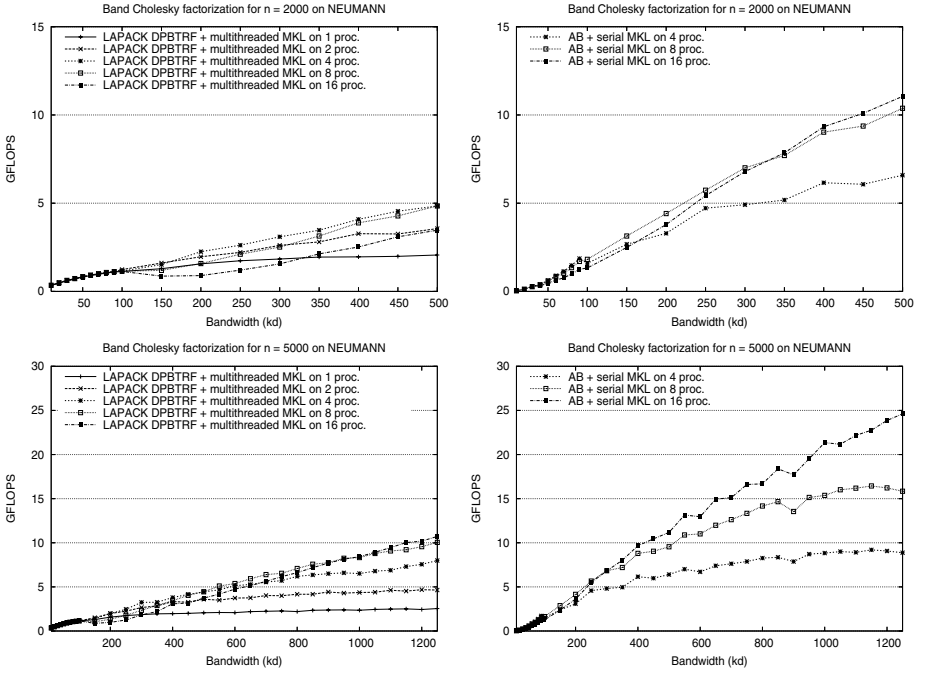


Fig. 4. Performance of the band Cholesky factorization algorithms on 1, 2, 4, 8, and 16 CPUs of NEUMMAN

We report the performance of two parallelizations of the Cholesky factorization:

- **LAPACK** DPBTRF **+ multithreaded MKL**. LAPACK 3.0 routine DPBTRF linked to multithreaded BLAS in MKL.
- **AB + serial MKL**. Our implementation of the algorithm-by-blocks linked to serial BLAS in MKL.

When hand-tuning block sizes, a best-effort was made to determine the best values of n_b in both cases.

Figures 3 and 4 report the performance of the two parallel implementations for band matrices of order $n = 2000$ and $n = 5000$ with varying dimension of the bandwidth and number of processors. The first thing to note from this experiment is the lack of scalability of the solution based on a multithreaded BLAS (plots on the left column): as more processors are added to the experiment, the left plots in the figure shows a notable drop in the performance so that using more than 2 or 4 processors basically yields no gain or even results in a performance decrease. The situation is different for the algorithm-by-blocks (plots on the right-hand side): For example, while using 4 or 8 processors on SET for a matrix of bandwidth below 200 attains a similar GFLOPS rate, using 8

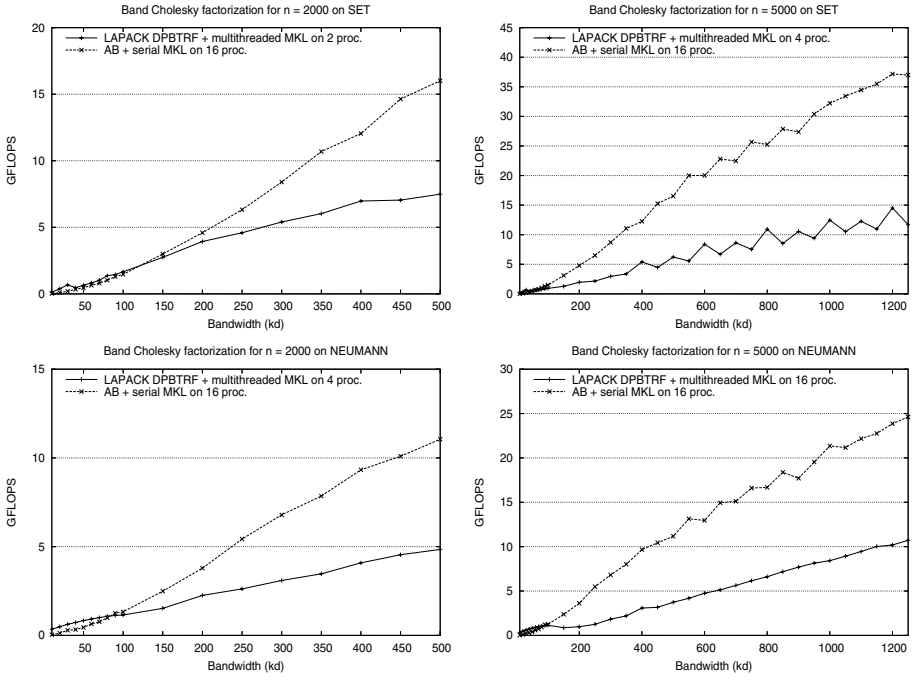


Fig. 5. Performance of the band Cholesky factorization algorithms

processors for matrices of larger bandwidth achieves a significant performance increase. A similar behavior occurs when all 16 processors of SET are employed but at a higher threshold, $k_d \approx 450$.

Figure 5 compares the two parallel implementations using the optimal number of processors: 2 (n=2000) and 4 (n=5000) on SET for the LAPACK DPBTRF+ multithreaded MKL implementation; 4 (n=2000) and 16 (n=5000) for this same algorithm on NEUMANN; and 16 for the AB + serial MKL implementation on both platforms. From this experiment it is clear the benefits of using an algorithm-by-blocks on a machine with a large number of processors.

6 Conclusions

We have presented an extension of SuperMatrix that yields algorithms-by-blocks for the Cholesky, LU (with and without pivoting) and QR factorizations of band matrices. The programming effort was greatly reduced by coding the algorithms with the FLAME/C and FLASH APIs. Using the algorithm-by-blocks, the SuperMatrix run-time system generates a DAG of operations which is then used to schedule out-of-order computation on blocks transparent to the programmer.

The results on two different parallel architectures for an algorithm-by-blocks for the band Cholesky factorization of matrices with medium to large

bandwidth clearly report higher performance and superior scalability to those of a traditional multithreaded approach using LAPACK.

Acknowledgments

We thank the other members of the FLAME team for their support. This research was partially sponsored by NSF grants CCF0540926 and CCF0702714. Gregorio Quintana-Ort, Enrique S. Quintana-Ort, and Alfredo Remon were supported by by the CICYT project TIN2005-09037-C02-02 and FEDER. This work was partially carried out when Alfredo Remn was visiting the Chemnitz University of Technology with a grant from the programme Plan 2007 de Pro- mocin de la Investigacin of the Universidad Jaime I.

We thank John Gilbert and Vikram Aggarwal from the University of California at Santa Barbara for granting the access to the neumann platform.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

References

1. Anderson, E., Bai, Z., Demmel, J., Dongarra, J.E., DuCroz, J., Greenbaum, A., Hammarling, S., McKenney, A.E., Ostrouchov, S., Sorensen, D.: LAPACK Users Guide. SIAM, Philadelphia (1992)
2. Bellens, P., Perez, J.M., Badia, R.M., Labarta, J.: CellSs: A programming model for the Cell BE architecture. In: SC 2006: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, p. 86. ACM Press, New York (2006)
3. Bientinesi, P., Gunnels, J.A., Myers, M.E., Quintana-Ortí, E.S., van de Geijn, R.A.: The science of deriving dense linear algebra algo- rithms. ACM Transactions on Mathematical Software 31(1), 1–26 (2005)
4. Bientinesi, P., Quintana-Ortí, E.S., van de Geijn, R.A.: Repre- senting linear algebra algorithms in code: The FLAME application programming interfaces. ACM Trans. Math. Soft. 31(1), 27–59 (2005)
5. Buttari, A., Langou, J., Kurzak, J., Dongarra, J.: A class of par- allel tiled linear algebra algorithms for multicore architectures. LAPACK Working Note 191 UT-CS-07-600, University of Tennessee (September 2007)
6. Buttari, A., Langou, J., Kurzak, J., Dongarra, J.: Parallel tiled qr factorization for multicore architectures. LAPACK Working Note 190 UT-CS- 07-598, University of Tennessee (July 2007)
7. Chan, E., Quintana-Ortí, E., Quintana-Ortí, G., van de Geijn, R.: SuperMatrix out-of-order scheduling of matrix operations for SMP and multi-core architectures. In: SPAA 2007: Proceedings of the Nineteenth ACM Sym- posium on Parallelism in Algorithms and Architectures, pp. 116–126 (2007)
8. Chan, E., Van Zee, F.G., Bientinesi, P., Quintana-Ortí, G., Quintana-Ortí, E.S., van de Geijn, R.: SuperMatrix: A multithreaded runtime scheduling system for algorithms-by-blocks. In: PPoPP 2008: Proceedings of the 13th ACM SIGPLAN symposium on Principles and practices of parallel programming. ACM Press, New York (to appear, 2008)

9. Chan, E., Van Zee, F., van de Geijn, R., Quintana-Ortí, E.S., Quintana-Ortí, G.: Satisfying your dependencies with SuperMatrix. In: IEEE Cluster 2007, pp. 92–99 (2007)
10. Chatterjee, S., Lebeck, A.R., Patnala, P.K., Thottethodi, M.: Recursive array layouts and fast matrix multiplication. IEEE Trans. on Parallel and Distributed Systems 13(11), 1105–1123 (2002)
11. Dongarra, J.J., Duff, I.S., Sorensen, D.C., van der Vorst, H.A.: Solving Linear Systems on Vector and Shared Memory Computers. SIAM, Philadelphia (1991)
12. Elmroth, E., Gustavson, F., Gustavson, F., Jonsson, I., Kagstrom, B.: Recursive blocked algorithms and hybrid data structures for dense matrix library software. SIAM Review 46(1), 3–45 (2004)
13. Henry, G.: BLAS based on block data structures. Theory Center Technical Report CTC92TR89, Cornell University (February 1992)
14. Leiserson, C., Plaat, A.: Programming parallel applications in Cilk. SIAM News, SINEWS (1998)
15. Low, T.M., van de Geijn, R.: An API for manipulating matrices stored by blocks. Technical Report TR-2004-15, Department of Computer Sciences, The University of Texas at Austin (May 2004)
16. Park, N., Hong, B., Prasanna, V.K.: Tiling, block data layout, and memory hierarchy performance. IEEE Trans. on Parallel and Distributed Systems 14(7), 640–654 (2003)
17. Quintana-Ortí, G., Quintana-Ortí, E., Chan, E., Van Zee, F.G., van de Geijn, R.: Scheduling of QR factorization algorithms on SMP and multi-core architectures. In: 16th Euromicro Int. Conference on Parallel, Dis- tributed and Network-based Processing. IEEE, Los Alamitos (to appear, 2008)
18. Quintana-Ortí, G., Quintana-Ortí, E.S., Chan, E., van de Geijn, R., Van Zee, F.G.: Design and scheduling of an algorithm-by-blocks for the LU factorization on multithreaded architectures. FLAME Working Note #26 TR-07-50, The University of Texas at Austin, Department of Computer Sciences (September 2007)
19. Quintana-Ortí, G., Quintana-Ortí, E.S., Remón, A., van de Geijn, R.: SuperMatrix for the factorization of band matrices. FLAMEWorking Note #27 TR-07-51, The University of Texas at Austin, Department of Computer Sciences (September 2007)
20. Remón, A., Quintana-Ortí, E.S., Quintana-Ortí, G.: Cholesky factorization of band matrices using multithreaded BLAS. In: Kågström, B., Elmroth, E., Dongarra, J., Waśniewski, J. (eds.) PARA 2006. LNCS, vol. 4699, pp. 608–616. Springer, Heidelberg (to appear, 2007)

An Efficient and Robust Decentralized Algorithm for Detecting the Global Convergence in Asynchronous Iterative Algorithms

Jacques M. Bahi[1], Sylvain Contassot-Vivier[2], and Raphaël Couturier[1]

[1] LIFC, University of Franche-Comte, Belfort, France
{jacques.bahi,raphael.couturier}@univ-fcomte.fr,
http://info.iut-bm.univ-fcomte.fr/
[2] LORIA, University Henri Poincaré, Nancy, France
sylvain.contassotvivier@loria.fr
http://www.loria.fr/~contasss/homeE.html

Abstract. In this paper we present a practical, efficient and robust algorithm for detecting the global convergence in any asynchronous iterative process. A proven theoretical version, together with a first practical version, was presented in [1]. However, the main drawback of that first practical version was to require the determination of the maximal communication time between any couple of nodes in the system during the entire iterative process. The version presented in this paper does not require any additional information on the parallel system while always ensuring correct detections.

Keyword: Asynchronous iterative algorithms, convergence detection.

1 Introduction

Iterative algorithms are very well suited to numerous problems in the context of scientific computations. They are often opposed to direct algorithms which give the exact solution of a problem within a finite number of operations whereas iterative algorithms provide successive approximations of it. It is said that they converge (asymptotically) towards the solution. When dealing with very large size problems, iterative algorithms are preferred, especially if they give a good approximation in a little number of iterations [2]. In other cases, they represent the only way to solve the problem as, for example, in the polynomial roots finding problem.

For all those reasons, parallel iterative algorithms are very popular. Nevertheless, most of them are synchronous and we showed in [3] that using asynchronism in such parallel iterative algorithms was far more efficient in the emerging contexts of parallelism such as grid computing. In the scope of this study, the term asynchronism means that each processor performs its local computations without waiting for the last updates of data dependencies coming from other processors. The asynchronous algorithms proposed in our first works used a centralized method to detect the global convergence, which was not best suited to grid contexts. In [1], both a theoretical and a practical version of a

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 240–254, 2008.

decentralized global convergence detection algorithm were proposed. The validity of the theoretical version designed for totally asynchronous algorithms was proven, and a practical version working on partially asynchronous algorithms, i.e. asynchronous algorithms with bounded delays, was deduced. However, that last algorithm presented the drawback of requiring additional information on the parallel system which is not easy, if not to say impossible, to collect in practice.

In this paper, we present another practical version of the decentralized algorithm for detecting global convergence that no longer requires any additional information on the system but only the local states of the nodes.

As in the previous version, messages for the computation and messages for the convergence detection are distinguished. This provides some degree of loss tolerance on the computational messages. The messages involved in the convergence detection are quite small and their limited number avoids any representative network overload which could penalize the progress of the iterative process. Finally, the delay of detection after the actual convergence stays reduced compared to the global process.

The following section briefly presents the previous studies related to convergence detection algorithms. Then, in order to be self-contained, the principles of asynchronous iterative algorithms are given in Sect. 3. Section 4 describes the main problems related to the convergence detection and the weaknesses of our previous algorithm. The new practical version of the decentralized algorithm for convergence detection is described in Sect. 5 and evaluated in Sect. 6.

2 Related Works

Most of the previous studies on the convergence detection problem in parallel iterative algorithms (see for example [4]) are based on centralized and/or synchronous algorithms (typically a global reduction), which are neither suited to large scale and/or distant distributed computations nor to the decentralized nature of asynchronous iterative algorithms.

Concerning the specific studies related to asynchronous iterative algorithms, distributed convergence detection was firstly introduced in [5] under particular assumptions, such as the particular behavior of the nodes which have reached local convergence. Moreover, Savari and Bertsekas proposed another distributed version in [6] under rather restrictive hypotheses such as FIFO communications and with modifications of the iterative process itself in order to make it terminate in finite time. Other authors have studied implementations of asynchronous algorithms but always with centralized convergence detection [7].

As in [1], the algorithm presented in this paper is based on a leader election algorithm to manage the termination of asynchronous iterative algorithms in a decentralized way. However, contrary to the practical version presented in that previous paper, the presented practical algorithm does not require any other information apart from the local convergence states of the nodes.

For more information on the distinction between the theoretical and the practical versions of our convergence detection algorithm, and on the leader election protocol, the reader should refer to [1] and the references therein.

3 Asynchronous Iterative Algorithms

Iterative algorithms have the structure $x^{k+1} = g(x^k)$, $k = 0, 1, ...$, with x^0 given, where each x^k is an n - dimensional vector, and g is some function from $\mathbb{R}^n$ into itself. A fixed point x^* of g is characterized by the property $g(x^*) = x^*$. The goal of the iterative algorithm is to reach such a fixed point starting from any initial vector x^0.

The parallel version of the iterative algorithm presented above is obtained by the classical block-decomposition of x into m block-components X_i, $i \in \{1, ..., m\}$, and g into a compatible way of m block-components G_i, to reformulate the iterative process as: $X_i^{k+1} = G_i\left(X_1^k, ..., X_m^k\right), i = 1, ..., m$, with X^0 given.

3.1 AIAC Algorithms

AIAC algorithms, which have been introduced in [8], are a variant of the totally asynchronous algorithms. The reader should refer to [9] and [10] to get the two major formulations of the theoretical model of totally asynchronous iterative algorithms. In this paper, we only remind the reader that those algorithms mainly induce the notion of delays between the components of the system. In totally asynchronous iterations, some classical conditions are assumed over those delays in order to ensure that the process actually evolves (see again [9]).

The acronym AIAC stands for Asynchronous Iterations - Asynchronous Communications. It means that all the processors perform their iterations without taking care of the progress of the other processors. They do not wait for predetermined data to become available from other processors but they keep on computing, trying to solve the given problem with whatever data happen to be available at that time. Those algorithms give very good results in the global context of grid computing as has been shown in [3]. Nevertheless, a centralized algorithm for detecting global convergence is not well suited to the context of grid computing in which all the nodes may not be directly accessible to each other for security reasons. Moreover, another reason for designing a decentralized convergence detection algorithm is that the most general class of parallel iterative algorithms corresponds to the asynchronous iterative algorithms, which are not centralized by nature.

4 Practical Difficulties with Our Previous Algorithm

The ideal way to detect the global convergence of an asynchronous iterative algorithm is to monitor the evolution of the global state of the system between two consecutive periods. In the field of dynamic systems, a period corresponds to a minimal span of time during which all the components of the system are updated at least once with data at least as recent as the beginning of that period. The evolution of the system is measured by the residual which is the distance between the two global states according to an adequately chosen norm. Hence, for any converging process, it has been shown that the residual using the adequate norm monotonously decreases from a period to the following one.

So, the convergence detection should only consist in verifying that this residual becomes small enough (under a convergence threshold).

However, it is quite difficult and penalizing in practice to identify periods at the global level of dynamical systems implemented on distributed systems. So, the method commonly used to detect the global convergence is based on a local notion of convergence. That local convergence is detected according to the local residual between two consecutive iterates of the local block-component according to the chosen norm. The local convergence is assumed when that residual becomes smaller than a given threshold. And the global convergence detection consists in verifying that all the nodes are in local convergence and that those local states have been reached with relevant updates of the respective dependent data. The major problem with this is that the local convergence is quite an artificial notion which is not directly linked to the global convergence. In particular, as the local residual is not computed on the global state-vector, it is subject to slightly less restrictive constraints and it may not monotonously decrease. So, additional mechanisms are required to avoid false detections.

In [1], each processor counts a given number of consecutive iterations for which its residual is under the convergence threshold and, only then, it passes in the local convergence state. Although in theory that number of consecutive iterations, which ensures the local convergence of the node, exists and is finite, it is quite difficult to evaluate in practice. Using an approximate value is possible and greatly reduces the potential false detections. However, it requires a global detection mechanism that takes into account possible divergences of the nodes.

In addition to that problem, which is common to every kind of iterative processes, the communication delays induced by the asynchronism change the behavior of the system and make it even more difficult to perform a correct convergence detection. Typically, a processor may be under the threshold thanks to old versions of data coming from some neighbor processors, which is not representative of its potential stabilization.

In Fig. 1 we illustrate such a false detection. We assume that a processor i has dependencies with processors $i-1$ and $i+1$. We also assume that there is a mechanism that detects the global convergence after a given span of time during which all processors are in the local convergence state. The convergence detection problem lies in the fact that processor 1 does not receive any message from its neighbor (processor 2) during several consecutive iterations. Consequently, its state reaches the local convergence that consequently enables the detection of a global convergence. Nevertheless, processor 1 only uses old messages from processor 2 to enter the local convergence state.

In [1], the global convergence detection mechanism is based on the leader election algorithm in order to obtain a decentralized algorithm. In this way, the local convergences of the nodes are propagated through a spanning tree of the system until they meet on a single node. Moreover, some canceling messages are used either to stop the propagation or to inform the elected node that one node is no longer in local convergence (due to the problem of delays exhibited above). This implies a waiting period after the global convergence detection

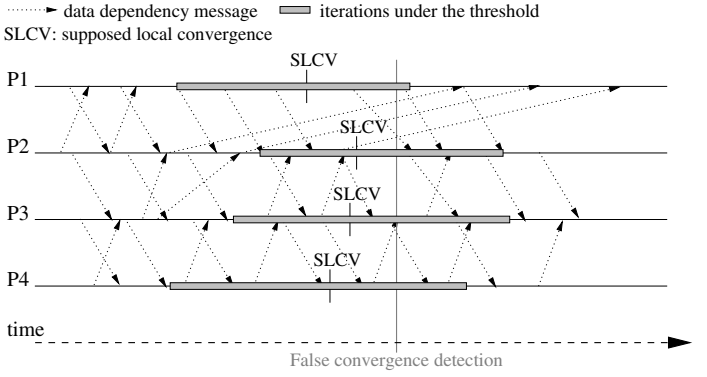


Fig. 1. Example of false convergence detection due to the fact that processor 1 does not receive messages from processor 2 during several consecutive iterations

on the elected node, in order to wait for any potential cancellation message generated in the meantime.

Although that overall detection algorithm works quite well in practice, a criticism can be formulated against it: two constants that depend on the system must be evaluated. The first one is the number of successive iterations under the threshold to assume the local convergence. It indirectly depends on the maximal delays between a processor and its dependencies, and consequently, on the computations performed in the iterative process. The second constant is the maximal traversal time of the system by a cancellation message. It obviously depends on the network configuration of the system. Such information is not easily evaluated accurately in practice.

To bypass those problems, we have designed a new decentralized detection algorithm that does not require any additional information about the system.

5 New Practical Version of the Decentralized Algorithm for Convergence Detection

As seen above, our major problem in the context of asynchronous algorithms is to get a correct image of the global state of the system. Indeed, the possible variations of the local states of the nodes require a robust snapshot of the global state of the system to ensure that all the nodes have verified the local convergence conditions at the same time.

The practical version presented here is somewhat different from the one proposed in [1]. Our new version does not require any specific information on the parallel system used. Our approach is closer to the theoretical version presented in our previous work in the sense that it lets the global detection happen even if the local evolutions on the nodes change during the election process. Then, after the global detection, an additional verification phase takes place to ensure its validity. It is important to notice here that the iterative process is not interrupted either during the global convergence detection process or during the

verification phase. There are two reasons for that; the most obvious one is not to slow down the iterative process itself, and the second one is that its evolution during the global detection and verification processes represents a mandatory piece of information.

The first of the two mechanisms mentioned above concerns the local convergence detection on each node and consists in taking into account what we call pseudo-periods in place of a given number (arbitrary in practice) of successive iterations. The pseudo-period is quite a local version of the periods. For each node, a pseudo-period corresponds to the minimal span of time during which the node receives at least one newer data message from all its dependencies and updates itself. In this way, the local evolution of the node is fully representative between two consecutive pseudo-periods. Thus, the local convergence is assumed only after at least one (but possibly several successive ones) pseudo-period is performed while the residual is under the threshold. This has a far better regulating effect on the local convergence detections in practice and, if it cannot avoid all the false local detections, it strongly limits them.

The second mechanism takes place at the global level of the system, when the global convergence is detected. As in our previous algorithm, the global detection is performed by a leader-election-like algorithm, according to the local convergences of the nodes. However, instead of using cancellation messages when the state of a node changes, as in our previous version, our new process lets the global detection occur. Nonetheless, a new step is added after that global detection which consists in verifying that all the nodes were actually in local convergence at the time of the detection and that their states were representative of their evolutions. That additional step is decomposed into four steps:

1. Diffusion of a verification message from the elected node through the spanning tree to initiate the verification phase;
2. Elaboration on each node of its response to the verification request;
3. Gathering of the responses of all the nodes toward the elected node through the spanning tree to get the verdict. The actual global convergence detection occurs at this step under the form of a detection confirmation;
4. Diffusion of a verdict message from the elected node through the spanning tree to finish the verification phase.

Some of those steps partially overlap in time. For example, when a node receives the verification message from one of its neighbors (the asking one), it forwards it to all its other neighbors (the replying ones) in the spanning tree (step 1) and, while waiting for their responses (step 3), it elaborates its own one (step 2) according to its local state. As soon as a negative response is detected on the node (either from itself or from a replying neighbor), the final response to the asking node can be sent. Otherwise, the node needs to wait for the gathering of all the responses from its replying neighbors before sending a positive response.

Finally, when the elected node has its own response and those of its neighbors, it deduces the final verdict, which corresponds to the actual global convergence detection when positive, and sends it to all of its neighbors (step 4). Then, each node receiving a verdict message forwards it to its other neighbors in the

spanning tree (step 4). At the end of the verification phase, the state of each node is set up according to that verdict.

As mentioned above, the response of each node depends on its state but also on its evolution during the verification phase. Indeed, in order to ensure that all the nodes are in local convergence at the same time (which corresponds to the criterion used in the sequential and synchronous versions), the response of a node is positive if and only if its residual stays under the threshold during the span of time between its last sending of a local convergence message (PartialCV) and the sending of its response to the verification request.

Moreover, to be sure that the response of each node is representative of its actual state and evolution, the waiting of a pseudo-period is inserted before the sending of the response. Hence, each node sends its response (depending on its residual evolution) only after having performed at least one iteration with versions of all its data dependencies at least as recent as the global detection time. In this way, the response is fully representative of the actual evolution and state of that node until that time. In fact, those pseudo-periods form, at the global level of the system, a period which spreads from the global convergence detection to the end of the pseudo-period on the latest node.

In order to force the nodes to use specific data versions during the verification phase, a tagging system is included in the data messages in order to differentiate them between the successive phases of the iterative process (normal processing and verification phase). Moreover, since there may be several verification phases during the whole iterative process, due to possible cancellations (negative verdicts) of global detections, that tagging is also useful to distinguish the data messages related to different verification phases.

Finally, such a tagging system is also useful in the messages related to the global detection and verification processes in order to enhance the reactivity of the verification phase. Indeed, as mentioned above, each node is allowed to send a negative response as soon as it is able to deduce it, without waiting for all the responses of its replying neighbors. It is also the case for the elected node that will send a negative verdict without waiting for all the responses of its neighbors. However, those unused responses must be correctly managed when they finally arrive on a node and, in particular, they must not be confused with other responses related to a more recent verification phase, as they may consequently overlap.

In order to respond to all those message distinction constraints, each phase of the iterative process (normal computing and verification of the global convergence) is distinguished in time by an integer tag incremented at each phase transition, as shown in Fig. 2 with four nodes linearly organized for a span of time beginning with the tag equal to k.

The whole mechanism of global detection and verification is detailed in Fig. 3, in the case of a global convergence detected and confirmed on node P_2. First of all, the processors reach local convergence and inform their adequate neighbor according to the leader election scheme, with PartialCV messages. Then, the global convergence is detected on node P_2 which initiates the verification phase by sending verification messages which are propagated through the system. As

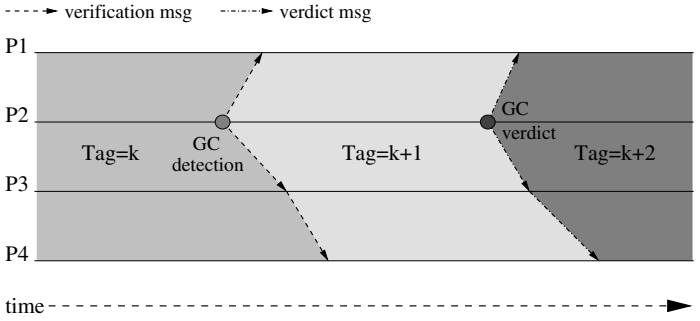


Fig. 2. Distinction of the successive phases during the iterative process

soon as the other nodes receive that message, they send their current version of local data to their neighbors with tagged data messages. And as soon as a node has received all its tagged data dependencies (not older than the last global detection) and has performed one iteration with them (dark grey blocks), it checks if its residual is still under the threshold since its last local convergence detection (light grey blocks) and sends the adequate response. As node P_3 is not elected and is not at an extremity of the system, it aggregates its response with the one of node P_4 and sends to its demanding node (P_2) the response corresponding to that sub-tree of the system relatively to the current root (P_2). The elected node P_2 meanwhile performs its own verification and as soon as it has finished it and has received all the other responses, it emits the verdict. The actual convergence detection takes place at this step when the verdict is positive. Finally, the verdict is sent and propagated through the system.

As can be seen, in case of a positive verdict, the whole process ensures that the residuals of all the nodes are under the threshold at least at the time at which the global convergence is detected on the elected node.

Concerning the correctness of the detection, we remind the reader that the ideal convergence criterion consists in verifying that the residual between two consecutive periods is small enough (under a convergence threshold). However,

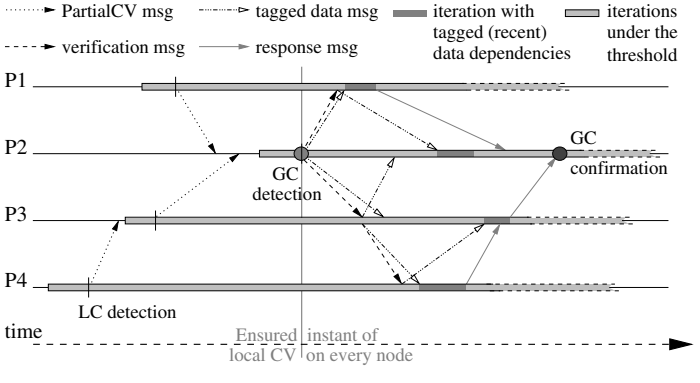


Fig. 3. Global convergence detection mechanism

as mentioned before, it is quite difficult and penalizing to identify all the periods at the global level during the process. But it is far simpler to explicitly trigger the execution of one period at a given time. This is what is done in Algorithm 9 in which the convergence criterion is composed of:

- A first pseudo-period with residual under the threshold;
- An arbitrary number (possibly 0) of consecutive pseudo-periods with residual under the threshold (the global convergence detection happens in that part);
- A last pseudo-period performed with data no older than the last global convergence detection and with residual under the threshold (the global convergence confirmation takes place at the end of that part).

The first two elements identify a global context of residual under the threshold. The last two elements contain an actual period which spreads between the global convergence detection and the global convergence confirmation. That period permits to ensure the validity of the detected convergence as it provides a similar stopping criterion as in the sequential and synchronous cases.

As the behavior of the nodes is not the same according to the different steps in the detection process and verification phase, it is also necessary to introduce four main states:

- **NORMAL:** the basic state during the iterative process when the node is not in the global convergence detection mechanism.
- **WAIT4V:** when the node is waiting for the local start of the verification phase after its sending of a PartialCV message.
- **VERIF:** when the node is performing the verification phase, either after the receipt of the corresponding message or by election.
- **FINISHED:** when the global convergence has been confirmed.

The transitions between those states are depicted in Fig. 4. Other states, present at inner levels (see Table 1), are not depicted here for clarity sake.

The final scheme obtained is given in Algorithm 9. Due to length constraints on that paper, only the list of the additional variables, according to the previous algorithm, is given in Table 1. The reader should refer to [1] for a description of the other variables.

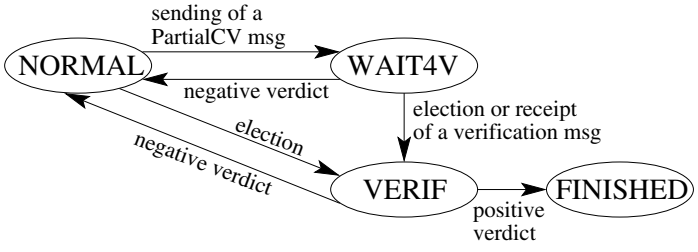


Fig. 4. State transitions in the global convergence detection process

Table 1. Description of the additional variables used in Algorithm 9

MyRank	unique identifier of the current node
State	current state of the node (NORMAL, WAIT4V, VERIF or FINISHED)
PhaseTag	identifier of the current phase on the node
PseudoPerBeg	boolean indicating that a pseudo-period has begun
PseudoPerEnd	boolean indicating the end of a pseudo-period
NbDep	number of computational dependencies of the node
NewerDep[NbDep]	boolean array indicating for each data dependency if a newer version has been received since the last pseudo-period
LastIter[NbDep]	integer array indicating for each dependency node the iteration of production of the last data received from that node
PartialCVSent	boolean indicating that a PartialCV message has been sent
ElectedNode	boolean indicating that the node is the elected one
Resps[NbNeig]	integer array containing the responses of the neighbors of the current node in the spanning tree. The values are either -1 (negative), 0 (no response yet) or 1 (positive)
ResponseSent	boolean indicating that the response has been sent

The different types of messages are listed below together with their contents:

- **data message:**
  - identifier of the source node
  - iteration number on the source node at the sending time
  - phase tag of the source node at the sending time
  - data
- **PartialCV message** and **verification message:**
  - identifier of the source node
  - phase tag of the source node at the sending time
- **response message:**
  - identifier of the source node
  - phase tag of the source node at the sending time
  - response of the source node
- **verdict message:**
  - identifier of the source node
  - new phase tag to use on the receiver
  - verdict

The algorithm also uses additional functions which are briefly described below:

- **InitializeState():** (Re-)initializes the variables related to the convergence detection process and sets the node in NORMAL state.
- **ReinitializePPer():** (Re-)initializes the variables related to the pseudo-period detection.
- **InitializeVerif():** Initializes a new verification phase.
- **RecvDataDependency():** Manages the receipts of data dependencies. That function takes into account any newer data when the receiver is not in verification phase. Otherwise, it filters the data produced after the last global convergence detection, that is to say, with the same phase tag as the receiver.

- **RecvPartialCV():** Manages the receipts of PartialCV messages. Also updates the local state of the node when an election is possible. However, a mutual exclusion mechanism is used to ensure that only one node is elected.
- **RecvVerification():** Manages the receipts of verification messages. The message is taken into account only when its phase tag corresponds to the following phase on the receiver.
- **RecvResponse():** Manages the receipts of response messages. The message is taken into account only when the phase tag in the message corresponds to the current phase tag on the receiver.
- **RecvVerdict():** Manages the receipt of the verdict on the non-elected nodes. The verdict is always taken into account and propagated through the spanning tree to set all the nodes either in FINISHED state or back in NORMAL state with a new phase tag. As no other global convergence detection can happen before the end of the propagation of the verdict, there cannot be any confusion with a similar message coming from a previous verification phase.
- **ChooseLeader(integer, integer):** Chooses the elected node when there are two possible candidates whose identifiers are given in parameters. That function is not detailed in the following since it directly depends on the referee policy used. The choice of that policy is quite free as its only constraint is to make a choice between the two proposed nodes.

Algorithm 1 Function InitializeState()
```
NbNotRecvd ← NbNeig
for Ind from 0 to NbNeig−1 do
  RecvdPCV[Ind] ← false
end for
ElectedNode ← false
LocalCV ← false
PartialCVSent ← false
ReinitializePPer()
State ← NORMAL
```

Algorithm 2 Function RecvVerification()
```
Extract SrcNode and SrcTag from the message
if SrcTag = PhaseTag + 1 then
  InitializeVerif()
  State ← VERIF
  Broadcast the verification message to all its
  neighbors but SrcNode
end if
```

Algorithm 3 Function InitializeVerif()
```
ReinitializePPer()
PhaseTag ← PhaseTag + 1
for Ind from 0 to NbNeig−1 do
  Resps[Ind] ← 0
end for
ResponseSent ← false
```

Algorithm 4 Function ReinitializePPer()
```
PseudoPerBeg ← false
PseudoPerEnd ← false
for Ind from 0 to NbDep−1 do
  NewerDep[Ind] ← false
end for
```

Algorithm 5 Function RecvDataDependency()
```
Extract SrcNode, SrcIter and SrcTag from the
message
SrcIndDep ← index of SrcNode in the list of
dependencies of the receiver (−1 if ∉)
if SrcIndDep ≥ 0 then
  if LastIter[SrcIndDep] < SrcIter and
  (State≠VERIF or SrcTag=PhaseTag) then
    Put the data from message at their place
    in the local data array used for the com-
    putations, according to SrcIndDep
    LastIter[SrcIndDep] ← SrcIter
    NewerDep[SrcIndDep] ← true
  end if
end if
```

Algorithm 6 Function RecvPartialCV()
```
Extract SrcNode and SrcTag from the message
SrcIndNeig ← index of SrcNode in the list of
neighbors of the receiver
if SrcIndNeig ≥ 0 and SrcTag = PhaseTag
then
  RecvdPCV[SrcIndNeig] ← true
  NbNotRecvd ← NbNotRecvd−1
  if NbNotRecvd=0 and PartialCVSent=true
    and ChooseLeader(MyRank, SrcNode)
        = MyRank then
    ElectedNode ← true
    InitializeVerif()
    Broadcast a verification message to all its
    neighbors
    State ← VERIF
  end if
end if
```

Algorithm 7 Function RecvResponse()

```
Extract SrcNode, SrcTag and SrcResp from the message
SrcIndNeig ← index of SrcNode in the list of neighbors of the receiver
if SrcIndNeig ≥ 0
  and PhaseTag = SrcTag then
  Resps[SrcIndNeig] ← SrcResp
end if
```

Algorithm 8 Function RecvVerdict()

```
Extract SrcNode, SrcTag and SrcVerdict from the message
if SrcVerdict is positive then
  State ← FINISHED
else
  InitializeState()
  PhaseTag ← SrcTag
end if
Broadcast the verdict message to all its neighbors but SrcNode
```

Algorithm 9 Decentralized algorithm for the global convergence detection

```
for all P_i, i ∈ {1, ..., N} do
  InitializeState()
  UnderTh ← false
  PhaseTag ← 0
  repeat
    ... iterative process, data sendings and evaluation of UnderTh ...
    if State = NORMAL then
      if UnderTh = false then
        ReinitializePPer()
      else
        if PseudoPerBeg = false then
          PseudoPerBeg ← true
        else
          if PseudoPerEnd = true then
            LocalCV ← true
            if NbNotRecvd = 0 then
              ElectedNode ← true
              InitializeVerif()
              Broadcast a verification message to all its neighbors
              State ← VERIF
            else
              if NbNotRecvd = 1 then
                Send a PartialCV message to the neighbor corresponding to the unique cell of RecvdPCV[] being false
                PartialCVSent ← true
                State ← WAIT4V
              end if
            end if
          else
            if all the cells of NewerDep[] are true then
              PseudoPerEnd ← true
            end if
          ...
      end if
    else if State = WAIT4V then
      if UnderTh = false then
        LocalCV ← false
      end if
    else if State = VERIF then
      if ElectedNode = true then
        if UnderTh = false or LocalCV = false
          or at least one cell of Resps[] is negative then
          PhaseTag ← PhaseTag + 1
          Broadcast a negative verdict message to all its neighbors
          InitializeState()
        else
          if PseudoPerEnd = true then
            if there are no more 0 in Resps[] then
              if all the cells of Resps[] are positive then
                Broadcast a positive verdict message to all its neighbors
                State ← FINISHED
              else
                PhaseTag ← PhaseTag + 1
                Broadcast a negative verdict message to all its neighbors
```

```
                InitializeState()
              end if
            end if
          else
            if all the cells of NewerDep[] are true then
              PseudoPerEnd ← true
            end if
          end if
        end if
      else
        if ResponseSent = false then
          if UnderTh = false or LocalCV = false
            or at least one cell of Resps[] is negative then
            Send a negative response to the asking neighbor
            ResponseSent ← true
          else
            if PseudoPerEnd = true then
              if there remains only one 0 in Resps[] then
                if the other cells of Resps[] are all positive then
                  Send a positive response to the asking neighbor
                else
                  Send a negative response to the asking neighbor
                end if
                ResponseSent ← true
              end if
            else
              if all the cells of NewerDep[] are true then
                PseudoPerEnd ← true
              end if
            ...
    end if
  until State = FINISHED
end for
```

6 Experiments

In order to evaluate the efficiency of our algorithm, we have compared it with our previous convergence detection algorithm on a typical asynchronous iterative algorithm based on the inverse power method. At each iteration of the algorithm, we solve a linear system using the multisplitting method [3]. A cluster of 16 machines (Pentium IV 3Ghz) with a 1Gbps network has been used. We have chosen the problem D of the CG problem, reported in [11], in which the matrix has a degree equals to 255,000 and 200 iterations are performed. That problem is very well suited to our comparison as it requires 200 convergence detections. We have implemented this algorithm in Java with the Jace environment [12]. In Table 2, we report the average times of ten executions of that problem with our new convergence detection algorithm and with our previous version. In the "without load" column, the machines run only our program without any other load. As can be seen, the execution times are very similar although slightly in favor of our new version. Also, in order to evaluate the robustness of our new algorithm, we have performed another series of experiments in the same conditions but we have slowed down some machines (2, 4 and 8) by adding an additional load on them. In order to obtain a correct convergence detection with our previous algorithm in such a context, some of its parameters, such

Table 2. Execution times with our two convergence detection algorithms and with or without external load

Version	Exec. times (s) without load	Exec. times (s) with 2 loads	Exec. times (s) with 4 loads	Exec. times (s) with 8 loads
Previous algorithm	661	740	772	866
New algorithm	655	712	732	809

as the number of successive iterations under the threshold and the maximal traversal time of the system, had to be increased. As can be seen in the last three columns of Table 2, such tunings imply larger detection latencies and thus worse execution times than our new version, which does not require any context-dependent tuning.

7 Conclusion

A new practical version of our decentralized algorithm for detecting the global convergence in asynchronous iterative algorithms has been proposed. That new version presents the advantage of not requiring any information on the parallel system employed. This strongly broadens the contexts of use of AIAC algorithms since they are then surely and efficiently usable with large scale parallel systems, such as grids, in which the communication delays are subject to sharp variations and their upper bound is difficult to evaluate.

References

1. Bahi, J., Contassot-Vivier, S., Couturier, R., Vernier, F.: A decentralized convergence detection algorithm for asynchronous parallel iterative algorithms. IEEE Transactions on Parallel and Distributed Systems 16, 4–13 (2005)
2. Saad, Y.: Iterative methods for sparse linear systems, 2nd edn. SIAM, Philadelphia (2003)
3. Bahi, J.M., Contassot-Vivier, S., Couturier, R.: Parallel Iterative Algorithms: from sequential to grid computing. Numerical Analysis & Scientific Computing Series. Chapman & Hall/CRC, Boca Raton (2007)
4. Maillard, N., Daoudi, E.M., Manneback, P., Roch, J.L.: Contrôle amorti des synchronisations pour le test d'arrêt des méthodes itératives. In: Renpar 14, Hamamet, Tunisie, pp. 177–182 (2002)
5. Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods. Prentice Hall, Engelwood Cliffs (1989)
6. Savari, S.A., Bertsekas, D.P.: Finite termination of asynchronous iterative algorithms. Parallel Computing 22, 39–56 (1996)
7. Charão, A.S.: Multiprogrammation parallèle générique des méthodes de décomposition de domaine. PhD thesis, INPG (2001)

8. Bahi, J., Contassot-Vivier, S., Couturier, R.: Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms. IEEE Transactions on Parallel and Distributed Systems 16, 289–299 (2005)
9. Bertsekas, D.P., Tsitsiklis, J.N.: Parallel and Distributed Computation: Numerical Methods. Prentice Hall, Englewood Cliffs (1989)
10. Tarazi, M.E.: Some convergence results for asynchronous algorithms. Numer. Math. 39, 325–340 (1982)
11. The NAS parallel benchmark (1996), `science.nas.nasa.gov/Software/NPB/`
12. Bahi, J., Domas, S., Mazouzi, K.: Jace: a java environment for distributed asynchronous iterative computations. In: 12th Euromicro Conference on Parallel, Distributed and Network based Processing, PDP 2004, Coruna, Spain, pp. 350–357. IEEE Computer Society Press, Los Alamitos (2004)

A Parallel Implementation of the Trace Minimization Eigensolver

Eloy Romero and Jose E. Roman

Instituto ITACA, Universidad Politécnica de Valencia,
Camino de Vera, s/n, 46022 Valencia, Spain
Tel.: +34-963877356, Fax: +34-963877359
{eromero,jroman}@itaca.upv.es

Abstract. In this paper we describe a parallel implementation of the trace minimization method for symmetric generalized eigenvalue problems proposed by Sameh and Wisniewski. The implementation includes several techniques proposed in a later article of Sameh, such as multi-shifting, preconditioning and adaptive inner solver termination, which accelerate the method and make it much more robust. A Davidson-type variant of the algorithm has been also considered. The different methods are analyzed in terms of sequential and parallel efficiency.

Topics: Numerical algorithms for CS&E, parallel and distributed computing.

1 Introduction

Let A and B be large, sparse, symmetric (or Hermitian) matrices of order n. We are concerned with the partial solution of the generalized eigenvalue problem defined by these matrices, that is, the computation of a few pairs (λ, x) that satisfy

$$Ax = \lambda Bx \ , \tag{1}$$

where λ is a real scalar called the eigenvalue and x is an n-vector called the eigenvector. This problem arises in many scientific and engineering areas such as structural dynamics, electrical networks, quantum chemistry, and control theory. In this work, we focus on the particular case that the wanted part of the spectrum corresponds to the smallest eigenvalues. Also, we are mainly concerned with matrix pairs where B is positive (semi-)definite, although this condition can be relaxed under some circumstances.

Many different methods have been proposed for solving the above problem, including subspace iteration, Krylov projection methods such as Lanczos or Krylov-Schur, and Davidson-type methods such as Jacobi-Davidson. Details of these methods can be found in [1,2,3,4]. Subspace iteration and Krylov methods perform best when computing largest eigenvalues, but usually fail to compute the smallest or interior eigenvalues. In that case, it can be useful to combine the method with a spectral transformation technique, that is, to solve

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 255–268, 2008.

$(A - \sigma B)^{-1}Bx = \theta x$ instead of Eq. 1. The problem with this approach is its high computational cost, since linear systems are to be solved at each iteration of the eigensolver, and they have to be solved very accurately. Preconditioned eigensolvers such as Jacobi-Davidson try to reduce the cost, by solving systems only approximately.

This work presents an implementation of the trace minimization eigensolver with several optimizations, including dynamic multishifting, approximate solution of the inner system and preconditioning. This method, described in section 2, can be seen as an improvement on the subspace iteration method that is able to compute smallest eigenvalues naturally. It can also be derived as a Davidson-type algorithm.

The implementation is being integrated as a solver in SLEPc, the Scalable Library for Eigenvalue Problem Computations. SLEPc is a software library for the solution of large, sparse eigenvalue problems on parallel computers, developed by the authors and other colleagues. An overview is provided in section 3, together with implementation details concerning the new solver, including the optimizations (preconditioning, multishifting and adaptive inner solver termination).

The paper is completed with section 4 showing the parallel performance of the implementations and the impact of the optimizations, and section 5 with some conclusions.

2 Trace Minimization Eigensolver

The trace minimization method for solving symmetric generalized eigenvalue problems was proposed by Sameh and Wisniewski [5], and developed further in [6]. The main idea of the method is to improve the update step of subspace iteration for generalized eigensystems as explained below.

Let Eq. 1 be the order n generalized eigenproblem to solve, and assume that X_k is a B-orthogonal basis of an approximate eigenspace associated to the smallest p eigenvalues, being $1 \leq p \ll n$. In subspace iteration, the sequence of computed approximations X_k is generated by the recurrence

$$X_{k+1} = A^{-1}BX_k, \quad k \geq 0 \ , \tag{2}$$

where the initial solution X_0 is an $n \times p$ full rank matrix. During the process, B-orthogonality of the columns of X_k is explicitly enforced. It can be shown that X_k eventually spans a subspace that contains the required eigenvectors [4].

This procedure requires the solution of p linear systems of equations with coefficient matrix A at each step k (one system per each column of X_k). Moreover, this has to be done very accurately (otherwise the global convergence is compromised), which is significantly expensive. Trace minimization tries to overcome that difficulty by exploiting a property stated in the following theorem.

Theorem 1 (Sameh and Wisniewski [5]). *Let A and B be $n \times n$ real symmetric matrices, with positive definite B, and let $\mathcal{X}$ be the set of all $n \times p$ matrices X for which $X^TBX = I_p$, $1 \leq p \leq n$. Then*

$$\min_{X \in \mathcal{X}} \operatorname{tr}(X^T A X) = \sum_{i=1}^{p} \lambda_i \ , \tag{3}$$

where $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of $Ax = \lambda Bx$. The equality holds if and only if the columns of the matrix X, which achieves the minimum, span the eigenspace corresponding to the smallest p eigenvalues.

The trace of a square matrix, $\operatorname{tr}(\cdot)$, is defined as the sum of its diagonal elements, and it can be shown to be equal to the sum of its eigenvalues [4].

The trace minimization algorithm updates the current approximation X_k by subtracting a correction Δ_k that it obtained by solving the following constrained minimization problem,

$$\begin{aligned} &\text{minimize } \operatorname{tr}\left[(X_k - \Delta_k)^T A (X_k - \Delta_k)\right] , \\ &\text{subject to } X_k^T B \Delta_k = 0 \ . \end{aligned} \tag{4}$$

As shown in [5], X_{k+1} satisfies

$$\operatorname{tr}(X_{k+1}^T A X_{k+1}) \leq \operatorname{tr}(X_k^T A X_k) \ , \tag{5}$$

if X_{k+1} is a B-orthonormal basis of the subspace spanned by $X_k - \Delta_k$.

By the method of Lagrange multipliers, the solution of the minimization problem, Eq. 4, can be computed by solving a set of saddle-point systems of linear equations,

$$\begin{pmatrix} A & BX_k \\ X_k^T B & 0 \end{pmatrix} \begin{pmatrix} \Delta_k \\ L_k \end{pmatrix} = \begin{pmatrix} AX_k \\ 0 \end{pmatrix} \ , \tag{6}$$

being $2L_k$ the Lagrange multipliers.

Nevertheless, we are interested only in Δ_k, so Eq. 6 can be reduced further, as shown in [5], resulting in a set of constrained positive semi-definite systems,

$$(PAP)\Delta_k = PAX_k, \quad X_k^T B \Delta_k = 0 \ , \tag{7}$$

where P is the projector onto the orthogonal complement of BX_k,

$$P = I - BX_k (X_k^T B^2 X_k)^{-1} X_k^T B \ . \tag{8}$$

These systems can be solved by means of an iterative linear solver such as conjugate gradient (CG) or generalized minimal residual (GMRES) method, with a zero vector as initial solution so that the constraint is automatically satisfied.

It may seem that the projector of Eq. 8 will introduce an excessive overhead in the computation. However, in practice the overhead is not so high since p is normally small and the projector is applied implicitly, i.e., without building matrix P explicitly. Furthermore, PAP is better conditioned than A (as shown in [6, Th. 2.3]), thus reducing the required number of linear solver iterations.

Algorithm 1 summarizes the basic trace minimization method.

Algorithm 1 (Trace minimization)

Input: matrices A and B, number of desired eigenvalues p,
dimension of subspace $s \geq p$
Output: resulting eigenpairs

Choose an $n \times s$ full rank matrix V_1 such that $V_1^T B V_1 = I_s$
For $k = 1, 2, \ldots$
1. Compute $W_k \leftarrow AV_k$ and $H_k \leftarrow V_k^T W_k$.
2. Compute all eigenpairs of H_k, (Θ_k, Y_k).
3. Compute the Ritz vectors, $X_k \leftarrow V_k Y_k$.
4. Compute the residual vectors, $R_k \leftarrow W_k Y_k - B X_k \Theta_k$.
5. Test for convergence.
6. Solve the inner system of Eq. 7 approximately.
7. $V_{k+1} \leftarrow B$-orthonormalization of $X_k - \Delta_k$.

End for

In step 2 of Algorithm 1, the eigenvalues have to be arranged in ascending order and the eigenvectors need to be orthonormalized. Note that the search subspace has dimension s, which can be for instance twice the number of wanted eigenpairs. At the end of the computation, the first p diagonal elements of Θ_k contain the computed Ritz values, and the first p columns of X_k contain the corresponding Ritz vectors.

It can be shown that the columns of X_k in Algorithm 1 converge to the eigenvectors with an asymptotic rate bounded by λ_i/λ_{s+1} [6, Th. 2.2]. Convergence can also be proved under the assumption that the inner systems in Eq. 7 are solved inexactly [6, §3.1].

Davidson-type version. The main drawback of Algorithm 1, inherited from subspace iteration, is that the dimension of the subspace of approximants is constant throughout the computation. In [6], Sameh and Tong propose a variant with expanding subspaces, in which the number of columns of V_k grows, as well as the dimension of the projected matrix, H_k. This variant is related to the Generalized Davidson method, because it uses the preconditioned residuals for expanding the search subspace. These residuals will replace the right hand sides of the inner systems, Eq. 7. Integrating these two ideas with Algorithm 1 results in the Davidson-type trace minimization method, Algorithm 2.

Algorithm 2 (Davidson-type trace minimization)

Input: matrices A and B, number of desired eigenvalues p,
block size $s \geq p$, maximum subspace dimension $m \geq s$
Output: resulting eigenpairs

Choose an $n \times s$ full rank matrix V_1 such that $V_1^T B V_1 = I_s$
For $k = 1, 2, \ldots$
1. Compute $W_k \leftarrow AV_k$ and $H_k \leftarrow V_k^T W_k$.

2. Compute s eigenpairs of H_k, (Θ_k, Y_k).
3. Compute the Ritz vectors $X_k \leftarrow V_k Y_k$.
4. Compute the residuals $R_k \leftarrow W_k Y_k - B X_k \Theta_k$.
5. Test for convergence.
6. Solve the systems $[P(A - \sigma_{k,i} B)P]d_{k,i} = Pr_{k,i}$ s.t. $X_k^T B d_{k,i} = 0$.
7. If $dim(V_k) \leq m - s$
   then $V_{k+1} \leftarrow B$-orthonormalization of $[V_k, \Delta_k]$,
   else $V_{k+1} \leftarrow B$-orthonormalization of $[X_k, \Delta_k]$.

End for

In Algorithm 2, the number of columns of V_k is s initially, but grows as the iteration proceeds. The order of H_k grows accordingly, but only s eigenpairs are computed in step 2, and therefore the number of columns of X_k is constant. The other main difference with respect to Algorithm 1 is step 6, in which the system to be solved is different. Vectors $r_{k,i}$ and $d_{k,i}$ denote the ith column of R_k and Δ_k, respectively. The role of $\sigma_{k,i}$ and other considerations will be discussed in subsection 3.2. Finally, step 7 expands the working subspace, except if the maximum dimension has been reached, in which case V_{k+1} is set to have $2s$ columns only.

Apart from a much better behaviour in terms of convergence of Algorithm 2 with respect to Algorithm 1, there is another significant benefit, namely the reduction of the cost of enforcement of the constraint. The orthogonality requirement $X_k^T B d_{k,i} = 0$ is now an implicit deflation of the s Ritz vectors, and s can be much smaller than in the original algorithm. In fact, our implementation defaults to $s = p$ in Algorithm 2.

3 Implementation and Parallelization

The implementation has been developed in the context of the SLEPc library, being our intent to include the eigensolver in forthcoming releases.

3.1 Overview of SLEPc

SLEPc, the Scalable Library for Eigenvalue Problem Computations [7][1], is a software library for the solution of large, sparse eigenvalue and singular value problems on parallel computers. It can be used for the solution of problems formulated in either standard or generalized form, both Hermitian and non-Hermitian, with either real or complex arithmetic.

SLEPc provides a collection of eigensolvers including Krylov-Schur, Arnoldi, Lanczos, Subspace Iteration and Power/RQI. It also provides built-in support for different types of problems and spectral transformations such as the shift-and-invert technique.

SLEPc is built on top of PETSc (Portable, Extensible Toolkit for Scientific Computation, [8]), a parallel framework for the numerical solution of partial

[1] `http://www.grycap.upv.es/slepc/`

differential equations, whose approach is to encapsulate mathematical algorithms using object-oriented programming techniques in order to be able to manage the complexity of efficient numerical message-passing codes. All the PETSc software is freely available and used around the world in many application areas.

PETSc is object-oriented in the sense that all the code is built around a set of data structures and algorithmic objects. The application programmer works directly with these objects rather than concentrating on the underlying data structures. The three basic abstract data objects are index sets, vectors and matrices. Built on top of this foundation are various classes of solver objects, including linear, nonlinear and time-stepping solvers. Many different iterative linear solvers are provided, including CG and GMRES, together with various preconditioners such as Jacobi or Incomplete Cholesky. PETSc has also the provision to interface with third-party software such as HYPRE.

SLEPc extends PETSc with all the functionality necessary for the solution of eigenvalue problems. SLEPc inherits all the good properties of PETSc, including portability, scalability, efficiency and flexibility. SLEPc also leverages well-established eigensolver packages such as ARPACK, integrating them seamlessly. Some of the outstanding features of SLEPc are the following:

- Easy programming with PETSc's object-oriented style.
- Data-structure neutral implementation.
- Run-time flexibility, giving full control over the solution process.
- Portability to a wide range of parallel platforms.
- Usable from code written in C, C++ and Fortran.

3.2 Implementation Details

The current prototype implementation incorporates several improvements described in [6]. Most of them refer to the solution of the linear system in step 6 in both algorithms, which is the most expensive operation.

Shifting Strategy. An important acceleration in the original algorithm, that has been incorporated also in the Davidson-type version, comes from *shifts*. Instead of Eq. 7, this technique solves the systems

$$[P(A-\sigma_{k,i}B)P]d_{k,i} = PAx_{k,i}, \quad X_k^T B d_{k,i} = 0 \ , \tag{9}$$

where $d_{k,i}$ and $x_{k,i}$ are the ith columns of Δ_k and X_k, respectively, and $\sigma_{k,i}$ is the associated shift at step k.

If the desired eigenvalues are poorly separated from the remaining part of the spectrum, the unshifted method converges too slowly. Choosing an appropriate value of $\sigma_{k,i}$ can improve the separation of eigenvalues and accelerate the convergence. The shift heuristic strategy implemented in both algorithms is detailed in [6]. This technique of *multiple dynamic shifts* consists in computing a different shift $\sigma_{k,i}$ for each required eigenvalue in every outer iteration. The heuristic can be summarized as follows (i_0 is the number of converged eigenpairs):

– For $i_0 + 1$:
  if $\theta_{i_0+1} + \|r_{i_0+1}\|_{B^{-1}} \leq \theta_{i_0+2} - \|r_{i_0+2}\|_{B^{-1}}$ (test for cluster)
    then $\sigma_{i_0+1} \leftarrow \theta_{i_0+1}$,
    else $\sigma_{i_0+1} \leftarrow \max\{\theta_{i_0+1} - \|r_{i_0+1}\|_{B^{-1}}, \lambda_{i_0}\}$.
– For every $j > i_0 + 1$:
  if $\sigma_{j-1} = \theta_{j-1}$ and $\theta_j < \theta_{j+1} - \|r_{j+1}\|_{B^{-1}}$
    then $\sigma_j \leftarrow \theta_j$,
    else $\sigma_j \leftarrow \max\{\theta_l : \theta_l < \theta_j - \|r_j\|_{B^{-1}}\} \cup \theta_{i_0+1}$.

For practical reasons, employing 2-norms instead of B^{-1}-norms is recommended in [6]. However, this simplification does not result in a safe lower bound for some B matrices with $\lambda_{max}(B) > 1$, because a vector r satisfying $\theta_j - \|r\|_{B^{-1}} \leq \theta_j - \|r\|_2$ exists if $\lambda_{min}(B^{-1}) < 1$, due to the property

$$\|x\|_2 \sqrt{\lambda_{min}(B^{-1})} \leq \|x\|_{B^{-1}} \leq \|x\|_2 \sqrt{\lambda_{max}(B^{-1})} \ . \tag{10}$$

To overcome this difficulty, our implementation can make use of an approximation of the smallest eigenvalue of B (provided by the user or computed by the Gershgorin circle theorem) to obtain a better estimation of the shifts, because

$$\|r\|_{B^{-1}} = \sqrt{r^T B^{-1} r} = \|r\|_2 \sqrt{z^T B^{-1} z} \leq \|r\|_2 \sqrt{\lambda_{max}(B^{-1})} = \|r\|_2 \lambda_{min}(B)^{-\frac{1}{2}}.$$

Linear Solver and Preconditioning. When shifting strategies are used, the inner system of Eq. 9 may be indefinite or ill-conditioned. In order to make its resolution feasible with an iterative solver, it is necessary to use an effective preconditioner. In PETSc, it is very easy to precondition a linear system whose coefficient matrix is available explicitly. However, the preconditioning of Eq. 9 is not trivial.

Eq. 9 is a projected linear system with an orthogonality constraint, much like the Jacobi-Davidson correction equation. In [9], Sleijpen et al. describe how to solve

$$(I - uu^T)(A - \sigma B)(I - uu^T)t = r, \quad \text{s.t.} \quad t \perp u,$$

using a Krylov solver and an approximation K of $A - \sigma B$. Adapting this idea to the context of trace minimization results in solving Eq. 9 using a Krylov solver with a left preconditioner K for $A - \sigma_{k,i} B$, the operator $(PKP)P(A - \sigma_{k,i} B)P$ and right hand side vector $(PKP)(PA)x_{k,i}$. The transformation

$$F = \left[I - K^{-1} BX (X^T B K^{-1} B X)^{-1} X^T B \right] K^{-1}$$

is employed to optimize the application of the operator because

– if $z = Fy$ and $Py = y$, then $Pz = z$ and $PKPz = y$; and
– if the initial solution fed into the Krylov solver, v_0, belongs to the range of the operator, then all the subsequently generated vectors are in that subspace.

As a result, the matrix-vector product $z = (PKP)P(A - \sigma_{k,i} B)Pv$ in the Krylov solver can be calculated as $z = F(A - \sigma_{k,i} B)v$.

The F transformation is precomputed as

$$F = \left[I - K^{-1}BX(X^TBK^{-1}BX)^{-1}X^TB\right]K^{-1} \tag{11}$$
$$= K^{-1} - K^{-1}BX(X^TBK^{-1}BX)^{-1}X^TBK^{-1} \tag{12}$$
$$= K^{-1} - \tilde{J}\tilde{J}^T, \tag{13}$$

where $\tilde{J} = K^{-1}J$, $BX = JR$, and $J^TK^{-1}J = I$. In the implementation, in each external iteration a K^{-1}-orthonormal basis of BX is built and then premultiplied by K^{-1}. Consequently, the product by F only requires an application of $\tilde{J}\tilde{J}^T$ and K^{-1} on the vector, together with a subtraction.

Stopping Criterion. Another issue that has a significant impact on overall performance is the stopping criterion for the inner system solver. We adapt the strategy proposed in [5] to the shifted inner system, Eq. 9, as developed in [6]. Like preconditioning, monitoring the convergence of the linear system can avoid awkward breakdowns and unnecessary iterations.

The linear solver is configured to stop when either the error estimate is less than a certain tolerance, $\tau_{k,i}$, or the iterations of the method exceed a certain maximum. The tolerances are calculated as

$$\tau_{k,i} = \begin{cases} \sqrt{tol} & \text{if } k = 1 \\ (\theta_{k,i} - \sigma_{k,i})/(\theta_{k-1,s} - \sigma_{k,i}), & \text{if } k > 1 \text{ and } \theta_{k,i} \neq \sigma_{k,i} \\ (\theta_{k-1,i} - \sigma_{k,i})/(\theta_{k-1,s} - \sigma_{k,i}), & \text{if } k > 1 \text{ and } \theta_{k,i} = \sigma_{k,i}, \end{cases} \tag{14}$$

where *tol* is the tolerance demanded to the eigensolver. In some cases, $\tau_{k,i}$ is too close to 1, resulting in a poor update of the associated eigenvector that slows down its converge. To avoid this, our implementation allows the user to specify a maximum value of the tolerance.

The maximum number of allowed iterations is also set by the user, since the optimal value is dependent on the preconditioner and the conditioning of the inner system. However too small values may prevent convergence of the method.

Orthogonalization. In terms of optimizations for parallel efficiency, the implementation makes use of an efficient Gram-Schmidt orthogonalization procedure available in SLEPc and explained in [10]. The default option is set up to Classical Gram-Schmidt with selective refinement.

4 Performance Analysis

This section summarizes the experiments carried out in order to evaluate the parallel performance of the implementations and the impact of the optimizations in the global convergence. The matrices used for the tests were taken from the Matrix Market and the University of Florida Sparse Matrix Collection. All test cases correspond to real symmetric generalized eigenproblems arising in real applications. The test cases used in the analysis of the various optimizations are listed in Table 1. For parallel performance tests, larger matrices were used, see Table 2.

Table 1. Test cases from Matrix Market for testing optimizations (p: positive definite, s: positive semi-definite, i: indefinite). The column *speed-up* shows the gain factor of Algorithm 2 with respect to Algorithm 1. The dash (–) indicates cases that cannot be solved with Algorithm 1.

		A		B		
	size	nonzeros	definiteness	nonzeros	definiteness	speed-up
BCSST02	66	2,211	p	66	p	0.81
BCSST08	1,074	7,017	p	1,074	p	2.80
BCSST09	1,083	9,760	p	1,083	p	2.21
BCSST10	1,086	11,578	p	11,589	s	1.31
BCSST11	1,473	17,857	p	1,473	p	0.90
BCSST12	1,473	17,857	p	10,566	s	1.54
BCSST13	2,003	42,943	p	11,973	s	1.47
BCSST19	817	3,835	i	817	p	–
BCSST21	3,600	15,100	i	3,600	p	–
BCSST22	138	417	i	138	p	–
BCSST23	3,134	24,156	i	3,134	p	–
BCSST24	3,562	81,736	i	3,562	p	–
BCSST25	15,439	133,840	i	15,439	p	–
BCSST26	1,922	16,129	i	1,922	p	–
BCSST27	1,224	28,675	p	28,675	i	1.26
BCSST38	8,032	355,460	p	10,485	i	–

Table 2. Test cases from UF Matrix Collection for analyzing parallel performance

	size	A nonzeros	B nonzeros
DIAMON5	19,200	9,347,698	3,115,256
BS01	127,224	6,715,152	2,238,384
GYRO	17,361	1,021,159	340,431

The tests were executed on two clusters: *Odin*, made up of 55 bi-processor nodes with 2.80 GHz Pentium Xeon processors, arranged in a 2D torus topology with SCI interconnect, and *MareNostrum*, consisting of 2,560 JS21 blade computing nodes, each with 2 dual-core IBM 64-bit PowerPC 970MP processors running at 2.3 GHz, interconnected with a low latency Myrinet network. In this machine, only 256 processors were employed due to account limitations.

Although in [6] CG is used as the solver for the linear system, in our experience only a few simple problems can be solved using that configuration. Unless otherwise stated, all tests reported in this section use GMRES with an algebraic multigrid preconditioner. This preconditioner is one of those provided by HYPRE [11]. With that configuration, all test cases in Tables 1 and 2 are solved without convergence problems.

Original Version Vs. Davidson-type Version. As it was pointed out previously, Algorithm 2 not only reduces the time spent in the inner iteration, but

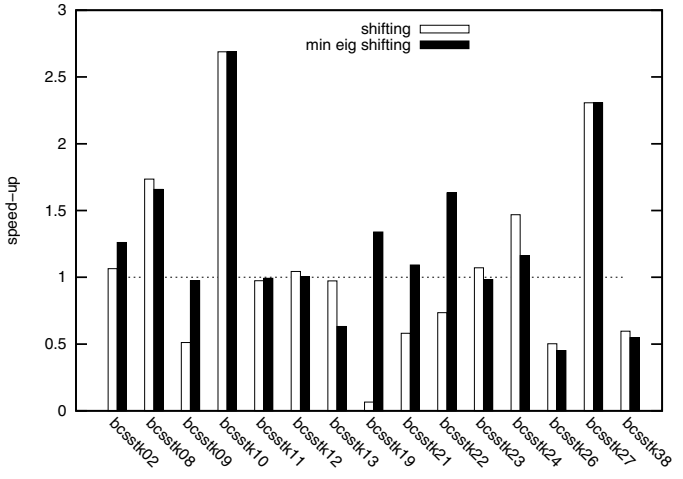


Fig. 1. Gain factor of dynamic shifting (`shifting`) and dynamic shifting with corrected norm (`min eig shifting`) with respect to the version without optimizations, in terms of the total number of inner iterations

also the total number of inner iterations. In order to support that conclusion, both algorithms have been compared using the test cases listed in Table 1. The last column of this table shows the gain factor of Algorithm 2 with respect to Algorithm 1, except for those problems with one of the two matrices indefinite, which cannot be solved with for Algorithm 1. Furthermore, the frequency of use of the most expensive operations is very similar in both variants, so the parallel speed-up and scalability should be very similar (this can be confirmed in Figures 4 and 6). For these reasons, the optimization and parallelism analyses will focus on the Davidson-type version.

Analysis of Optimizations. The benefits of the different improvements described in section 3.2 have been analyzed using the test cases listed in Table 1. The results are shown in Figures 1, 2, and 3, as gain factors of the different strategies with respect to the version without optimizations, in terms of the total number of inner iterations. It can be observed from the figures that the optimizations are effective in most problems (but not all), making the computation up to 5 times faster in several cases, or even more.

The shifting strategy (Figure 1) accelerates the convergence in 66% of the problems. The version that employs a shifting strategy with the correction provided by the smallest eigenvalue of B (see section 3.2) is usually to be preferred, although it may be worse sometimes depending on the definiteness of matrix B. The tolerance strategy (Figure 2) presents better speed-up, specially setting its upper bound to 0.1. Finally, the combination of the two techniques is plotted in Figure 3.

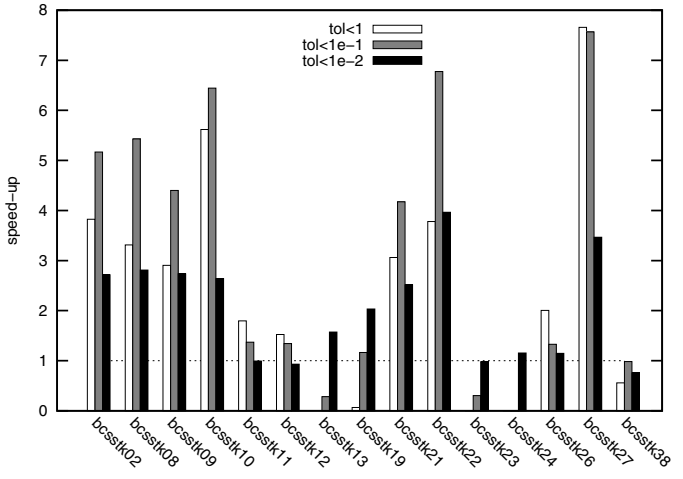


Fig. 2. Gain factor of the tolerance strategy with an upper bound of 1 (`tol<1`), 0.1 (`tol<1e-1`) and 0.01 (`tol<1e-2`), in terms of the total number of inner iterations

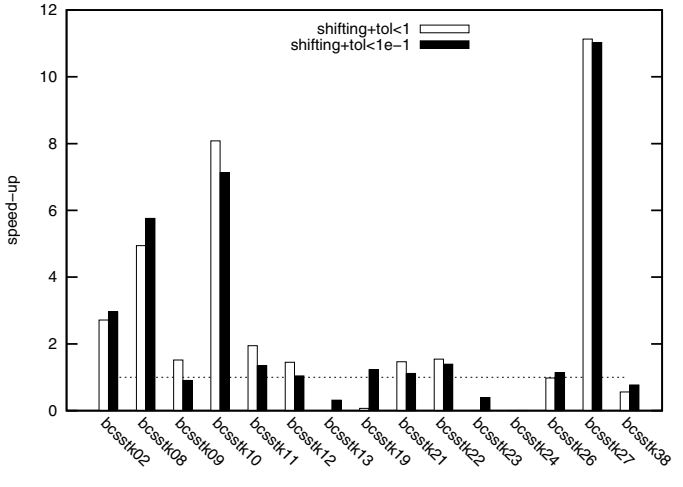


Fig. 3. Gain factor of dynamic shifting combined with tolerance strategy, in terms of the total number of inner iterations

Parallel Analysis. We start the parallel performance analysis with a scalability study, that is, to measure the parallel speed-up for different number of processors when the problem size is increased proportionally. Figure 4 presents the scalability results in both clusters for the Davidson-type trace minimization method, and also for the Krylov-Schur method (with shift-and-invert spectral transformation), using diagonally dominant random tridiagonal A and B matrices. In this case, the Jacobi preconditioner was employed when solving the inner systems.

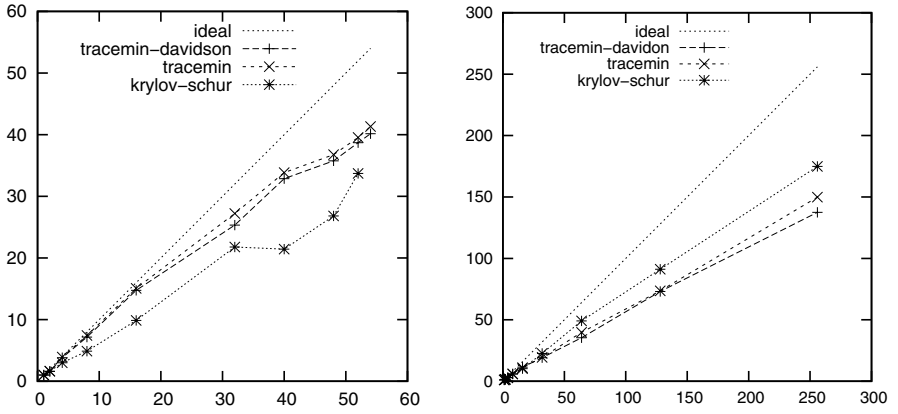


Fig. 4. Scalability of Davidson-type trace minimization and Krylov-Schur methods using a tridiagonal eigenproblem in odin (left) and MareNostrum (right)

The plots in Figure 4 indicate that both algorithms are reasonably scalable, considering the fact that they consist in a nested inner-outer iteration requiring a lot of communication among processors. In Odin, trace minimization seems to scale better than Krylov-Schur, but the roles are reversed in MareNostrum. This different behaviour could be explained by noticing that trace minimization spends more time in vector dot products (VecMDot) and norms (VecNorm) than in vector additions (VecMAXPY) and matrix-vector multiplications (MatMult), see Table 3. Vector products and norms scale worse, since they require a multi-reduction operation involving all processors, whereas VecMAXPY operations are trivially parallelizable and MatMult operations scale usually well. Maybe, the multi-reduction operation is less efficient in MareNostrum due to the hardware configuration.

We also presents results from some realistic problems, in particular those listed in Table 2. The speed-up of Algorithm 2 for all three problems is shown in Figure 5 for both platforms. For reference, the figure plots also the speed-up for the preconditioner application only, showing a strong correlation between the speed-up of the method and the preconditioner. We can conclude that a bad speed-up in the eigensolver can be attributed to a non-scalable preconditioner or a matrix with a disadvantageous sparsity pattern.

Finally, in Figure 6, we compare the speed-up of the Davidson-type implementation and the original trace minimization with the largest test case, showing a slight advantage of the former.

Table 3. Percentage of execution time corresponding to the three most time consuming operations, with 128 processors in MareNostrum

	VecMAXPY	MatMult	VecMDot	VecNorm
Krylov-Schur	32%	19%	10%	10%
Trace Minimization	29%	26%	24%	8%

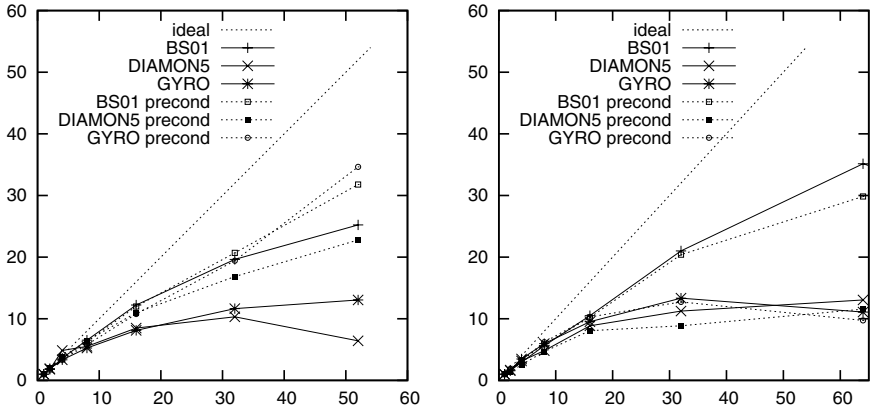


Fig. 5. Speed-up of Davidson-type trace minimization, as well as only the preconditioner application, for the BS01, DIAMON5, and GYRO problems in Odin (left) and MareNostrum (right)

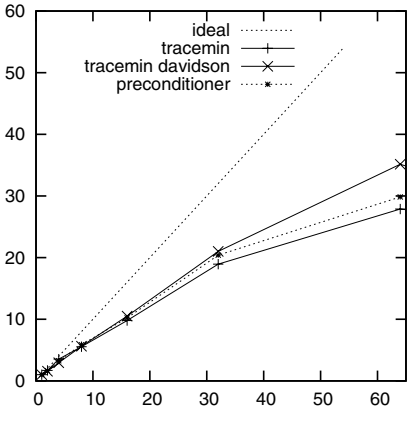


Fig. 6. Speed-up of the original trace minimization and Davidson-type methods, as well as only the preconditioner application, for the BS01 problem in MareNostrum

5 Conclusions

A parallel implementation of the trace minimization method for generalized symmetric eigenvalue problems has been developed and analyzed, focusing on its Davidson-type version. This implementation will be made available as a new solver in the SLEPc library. Our tests with several problems show that the proposed optimizations, such as multishifting, preconditioning and adaptive inner solver termination, accelerate the method considerably and make it more robust.

The parallel performance is comparable to that of Krylov-Schur, but the main advantage is that trace minimization can find the smallest eigenvalues in

problems where Krylov-Schur (without shift-and-invert) cannot. This development is the prelude to the implementation in SLEPc of more advanced preconditioned eigensolvers such as Jacobi-Davidson.

Acknowledgements. The authors thankfully acknowledge the computer resources and assistance provided by the Barcelona Supercomputing Center (BSC).

References

1. Parlett, B.N.: The Symmetric Eigenvalue Problem. Prentice-Hall, Englewood Cliffs (1980); Reissued with revisions by SIAM, Philadelphia (1998)
2. Saad, Y.: Numerical Methods for Large Eigenvalue Problems: Theory and Algorithms. John Wiley and Sons, New York (1992)
3. Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H. (eds.): Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, Philadelphia, PA. Society for Industrial and Applied Mathematics (2000)
4. Stewart, G.W.: Matrix Algorithms. Eigensystems, vol. II. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA (2001)
5. Sameh, A.H., Wisniewski, J.A.: A trace minimization algorithm for the generalized eigenvalue problem. SIAM J. Numer. Anal. 19(6), 1243–1259 (1982)
6. Sameh, A., Tong, Z.: The trace minimization method for the symmetric generalized eigenvalue problem. J. Comput. Appl. Math. 123(1-2), 155–175 (2000)
7. Hernandez, V., Roman, J.E., Vidal, V.: SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. ACM Trans. Math. Soft. 31(3), 351–362 (2005)
8. Balay, S., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc users manual. Technical Report ANL-95/11 - Revision 2.3.3, Argonne National Laboratory (2007)
9. Sleijpen, G.L.G., van der Vorst, H.A., Meijerink, E.: Efficient expansion of subspaces in the Jacobi–Davidson method for standard and generalized eigenproblems. Electron. Trans. Numer. Anal. 7, 75–89 (1998)
10. Hernandez, V., Roman, J.E., Tomas, A.: Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement. Parallel Comput. 33(7–8), 521–540 (2007)
11. Henson, V.E., Yang, U.M.: BoomerAMG: A parallel algebraic multigrid solver and preconditioner. Applied Numerical Mathematics: Transactions of IMACS 41(1), 155–177 (2002)
12. Sleijpen, G.L.G., der Vorst, H.A.V.: A Jacobi–Davidson iteration method for linear eigenvalue problems. SIAM Rev. 42(2), 267–293 (2000)

A Load Balancing Knapsack Algorithm for Parallel Fuzzy c-Means Cluster Analysis

Marta V. Modenesi, Alexandre G. Evsukoff, and Myrian C.A. Costa

COPPE/Federal University of Rio de Janeiro,
P.O.Box 68506, 21945-970 Rio de Janeiro RJ, Brazil
Tel.: (+55) 21 25627388, Fax: (+55) 21 25627392
marta.modenesi@petrobras.com.br,
alexandre.evsukoff@coc.ufrj.br, myrian@nacad.ufrj.br.

Abstract. This work proposes a load balance algorithm to parallel processing based on a variation of the classical knapsack problem. The problem considers the distribution of a set of partitions, defined by the number of clusters, over a set of processors attempting to achieve a minimal overall processing cost.

The work is an optimization for the parallel fuzzy c-means (FCM) clustering analysis algorithm proposed in a previous work composed by two distinct parts: the cluster analysis, properly said, using the FCM algorithm to calculate of clusters centers and the PBM index to evaluate partitions, and the load balance, which is modeled by the multiple knapsack problem and implemented through a heuristic that incorporates the restrictions related to cluster analysis in order to gives more efficiency to the parallel process.

Keywords: Unsupervised Classification, Fuzzy c-Means, Load Balance, Optimization.

1 Introduction

Cluster analysis is the unsupervised classification of data into groups (clusters) and it is one of the most intensive computational tasks in data mining. It is thus very attractive for parallel processing and many parallel and distributed clustering algorithms have been recently studied [1][2][3].

There are several approaches that have been studied for cluster analysis algorithms [4]. In the partition approach, two main optimization problems are addressed: to find the number of clusters presents in the data and the location of clusters centers. The later problem is much easier to solve, and iterative greedy optimization algorithms, such as the k-means algorithm and its variants, are widely used for that purpose, being well known by the data mining community.

The k-means algorithm has been extended to the fuzzy c-means algorithm by Bezdek in the early eighties [5]. The fuzzy c-means (FCM) algorithm computes a "fuzzy" partition where data records may be related to more than one cluster but with different membership values. The FCM solution is very useful in real applications because it provides soft boundaries for clusters taking classification uncertainty into account.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 269–279, 2008.

The problem of determining the number of clusters in a dataset is much more difficult to solve, both for crisp and fuzzy clustering algorithms. In general, a validation index that reflects the quality of the result is used as an optimization index and the clustering algorithm is executed for a range of clusters. Several cluster validation indexes have been proposed [6][7][8] and most of them are based on geometrical approaches with the aim of finding dense and separated clusters.

In a previous work, Modenesi et al. [3] have presented a parallel clustering algorithm that computes the location of clusters centers and the validation index simultaneously. In their approach, the FCM algorithm is iterated within the cluster validation loop, in which the clustering quality is computed by the PBM index, recently proposed by Pakhira et al. [8]. In the parallel implementation, the dataset is equally divided among the available processors, which compute the iterative steps, and the cluster and validation results are integrated by the master processor [3]. This approach causes a natural load balance in the parallel processing, but it has a high communication cost due to the need of frequent interaction among processors. The results show that parallelization is not efficient for a low number of clusters but the greater the dataset the better is the speed-up.

In cluster analysis, the question of minimizing communication costs and maximizing the parallel processing efficiency can be understood as a knapsack problem where computational capacity must be fulfilled minimizing the cluster analysis computation cost.

The knapsack problem, proposed initially by Dantzig [9], consists in the problem of selecting, from a collection of items, with distinct benefits and costs, the ones that fit in a knapsack resulting in the maximum possible value and minimal cost. It is a well known problem in the area of combinatorial analysis with extensive applicability in many practical cases, such as: production and logistics planning, financial engineering, vehicles shipment, budget, among others. The knapsack problem is a NP-complete problem and, therefore, there is no global optimum solution known to be computed in polynomial time. Many methods, called heuristics, are studied for solving the knapsack problem. They only give approximate solutions to the problem [10][11][12].

In this work, the parallel FCM cluster analysis proposed in a previous work [3] is extended with a load balancing computed by the solution of the knapsack problem. A heuristic algorithm is proposed, based on the bin packing problem, which is a variation of the multiple knapsacks problem.

The paper is organized as follows: section two reviews the parallel FCM cluster analysis algorithm; section three introduces the knapsack problem; section four presents the FCM cluster analysis with the load balance algorithm; section five describes tests with results and section six presents conclusions and suggests future works.

2 The Parallel FCM Algorithm

The cluster analysis FCM [3] aims to find the patterns present in data by processing a range of clusters by the calculation of distances of registers to clusters centers through the FCM algorithm and the selection of the best partition through the cluster validity index PBM.

The parallel FCM cluster analysis procedure is described by the following sequence:

Step 1. (Master processor): Splits the data set equally among the available processors so that each one receives N/p records, where N is the number of records and p is the number of processors.

Step 2. (All processors): Compute the geometrical center of its local data and communicate this center to all processors, so that every processor can compute the geometrical center of the entire database. Compute the global data density (factor E_1 of the PBM index) on local data and send it to master processor.

Step 3. (Master processor): Sets initial centers and broadcasts them, so that all processors have the same clusters' centers values at the beginning of the FCM looping.

Step 4. (All processors): Until convergence is achieved compute the distances from each record in the local dataset to all clusters' centers; update the partition matrix, calculate new clusters' centers.

Step 5. (All processors): Compute the cluster density (factor E_K of the PBM index) on its local data and send it to master processor.

Step 6. (Master Processor): Integrates the calculations for the PBM index and stores it. If the range of number of clusters is covered, stops, otherwise returns to *Step3*.

The procedure described above is computed for each number of clusters in the cluster analysis, so that the procedure is repeated as many times as the desired range of numbers of clusters, so that the PBM index, as a function of the number of centers, is computed. The best partition is the one corresponding to the largest value of the PBM index.

The computational cost of the algorithm is exponentially proportional to the number of patterns being analyzed. The bigger is the number of clusters to calculate, more cycles of processing of distances of registers to clusters centers are necessary and greater is the computational effort. Moreover, each partition has a different computational cost which is related to its number of clusters, meaning that processing partitions with bigger number of patterns (clusters) involve more computational effort than processing those with smaller ones.

An approach to minimize the communication cost in the prior algorithm is to distribute partitions over processors, making processors in charge of calculating a partitions´ group with no need of communication during the FCM loop, having at the end of the processing a master processor consolidating the results and indicating which is the best partition.

To make this approach effective, a load balance policy must be implemented, ensuring that processors receive a balanced charge of partitions to process in order to minimize the total time of the parallel processing. The load balance policy must distribute partitions over processors considering the computational cost of each partition. At this point the knapsack problem comes as a model of how to arrange partitions minimizing the processing cost.

3 The Knapsack Problem

All knapsack problems variations refer to a set of n items where each item has an associated profit p_j and weight $w_j, 1 \le j \le n$, which are generally positive integers.

The problem consists in selecting which items must be included into the knapsack so that the total value is maximized without exceeding the knapsack capacity W . The selection value of an item is represented by the binary project variable $x_j \in \{0,1\}$.

The knapsack problem in its basic form can be formulated as:

$$\begin{aligned} &Maximize \sum_{j=1}^{n} p_j x_j \\ &s.t. \ \sum_{j=1}^{n} w_j x_j \leq W \end{aligned} \tag{1}$$

Many kinds of knapsack problems arise from real situations where different constrains determine special cases. Among the knapsack problem direct generalizations, were applicable in this work the subset sum problem and the multiple knapsacks problem.

The subset sum problem is characterized by the situation where items costs (weights) are equal to profits (values), i.e. $w_j = p_j$ [11][12]. It occurs when a quantitative target should be reached, so that its negative deviation (or loss) must be minimized and a positive deviation is not allowed.

The knapsack problem becomes useful for load balancing in parallel processing when a set of m knapsacks are considered. This problem is known as the subset-sum partition problem when the capacities of all knapsacks are equal, so that each knapsack capacity is $W_i = W / m$, where W is the overall capacity. The solution to the problem is represented by the binary variable $x_{ij} \in \{0,1\}$ that assigns an item j to the knapsack i .

The multiple knapsacks' problem is formulated as:

$$\begin{aligned} &Maximize \sum_{i=1}^{m} \sum_{j=1}^{n} w_j x_{ij} \\ &s.t. \ \sum_{j=1}^{n} w_j x_{ij} \leq W_i, \ 1 \leq i \leq m \end{aligned} \tag{2}$$

A special case of the multiple knapsacks problem is the bin packing problem, where items must be placed in the smallest number of knapsacks.

The formulation of the multiple knapsacks problem for load balancing in the FCM parallel cluster analysis is discussed next.

4 The Proposed Algorithm

4.1 Problem Formulation

The load balancing problem of the parallel FCM cluster analysis algorithm can be understood as a knapsack problem whose items are partitions to be processed by the algorithm, each partition defined by the number of clusters in the fuzzy partitions

range $2 \le j \le n$. The load balancing problem is a multiple knapsacks problem where each processor represents a knapsack and the overall processing capacity is defined by the number of the available processors p.

The main issue in the load balancing problem is to minimize the overall parallel processing time. The problem thus becomes a minimization problem, stated as follows:

$$\begin{aligned} &Minimize \sum_{i=1}^{p} \sum_{j=2}^{n} w_{ij} x_{ij} \\ &s.t. \sum_{i=1}^{p} x_{ij} = 1,\ 2 \le j \le n \\ &\sum_{i=1}^{p} \sum_{j=2}^{n} x_{ij} = n \end{aligned} \tag{3}$$

The project variable $x_{ij} \in \{0,1\}$ selects processor i to evaluate the partition j, so that each partition is evaluated only once as it is expressed by the first constraint. The second constraint states that all partitions should be evaluated.

The load w_{ij} represents the processing cost to evaluate the partition j at processor i and depends mainly on the number of partitions itself, besides the number of records and variables of the dataset [3].

In this work, the efficiency results obtained in [3] were used to define the loads as:

$$w_{ij} = \frac{1}{\varepsilon_{ij}} \tag{4}$$

where ε_{ij} is the efficiency, i.e. the ratio between the speed-up and the number of processors, computed by the evaluation of the partition j on processor i.

4.2 Description of the Algorithm

To ensure the optimization of the parallel processing time the definition of the knapsacks size (processing lines) is a critical factor. In the context of the FCM cluster analysis all partitions must be processed and is reasonable to conclude the ideal load the average of the partitions values, where, being K_i a set of partitions where $2 \le i \le n$ and p the number of processors, the average processing load would be:

$$S = \frac{1}{p} \sum_{i=1}^{p} \sum_{j=2}^{n} w_{ij} \tag{5}$$

This average size, however, cannot be always obtained in an exact manner. To ensure an efficient load balance strategy scalable for any set of partitions and number of processors, this work proposes a heuristic that uses two values: a lower limit defined

by the partitions average (the ideal load for the distribution) and a superior limit, which is the maximum load in the distribution that indicates the least possible cost for the parallel processing.

The heuristic assemblies the knapsacks through an iterative process where a distribution is generated and evaluated by a variation coefficient passed as a parameter at the beginning of the parallel process. If the group of knapsacks generated achieves a good variation coefficient, the algorithm goes on processing the FCM cluster analysis. Otherwise, the initial average value S is increased. If the S adjusted value surpasses the distribution maximal load, the algorithm goes on processing the FCM cluster analysis. On the contrary, a new distribution is generated, in an attempt to push the loads to a smaller number of processors, trying to free processors that will be reallocated to the evaluation of the bigger loads in order to reduce the overall parallel processing time. The looping ends when the distribution evaluation is considered ok or when the lower limit gets equal to the superior limit.

The heuristic core is based on the well known first fit decreasing algorithm much used to treat the bin packing problem [13][14].

The parallel FCM cluster analysis balancing scheme is described by the following sequence:

Step1. (Master processor): **Initial values calculation**
Sort partitions in decreasing order
Calculate the average S as (5).

Step2. (Master processor): **Knapsacks generation**
2.1 Assign higher cost partitions
Place each partition with $w_{1j} > S$ into a single processor and consider the processor line full
Assign remaining partitions
For each partition j, place the partition in the next processor where lines sum >= partition cost
2.3 Assign partitions that did not fit in processing lines
Until all unassigned partitions are placed
Sort processing lines by load size
Place partition in the processor with the smaller total load
If it is the first distribution identify maximum load for the processing

Step3. (Master processor): **Knapsacks evaluation**
3.1 Calculate average, standard deviation and the load variation coefficient
If variation coefficient > input variation
3.2 Adjust distribution parameters
a. $S = S + 1$
b. Cancel processors with maximum load with idle processors to find out the iteration maximum load
3.3 Evaluate if knapsack can be reorganized

a. If S < iteration maximal load
b. Returns to Step 2

Step4. (Master processor): **Assign idle processors**
If there is any idle processor
For each idle processor, assign to processor the next greatest load

Step5. (Master processor): **Communicate to processors**
Communicate processors´ partition lines
Communicate group information to processors who are part of a group

Step6. (Master processor): **Knapsack processing** *(All processors)*
6.1 If processor belongs to a group
Split data among the group processors
6.2 For each partition in processor calculate FCM and PBM loop
6.3 Send results to master processor

Step7. (Master processor): **Select partition with the greatest PBM index**

When the rate between number of partitions and number of processors is high the load balancing generated is usually a good one and presents very small variation coefficient (Fig. 1). When the distribution using the average does not achieve a good result, such as in cases where the number of partitions is very close to the number of processors or, when the processors numbers are bigger than the number of partitions, the algorithm recalculates the knapsacks distribution, "pushing" the loads to a point, trying to group them in fewer processors, so that some processors can be freed to process along others the higher cost loads in an attempt of improve de overall load balance (Fig. 2).

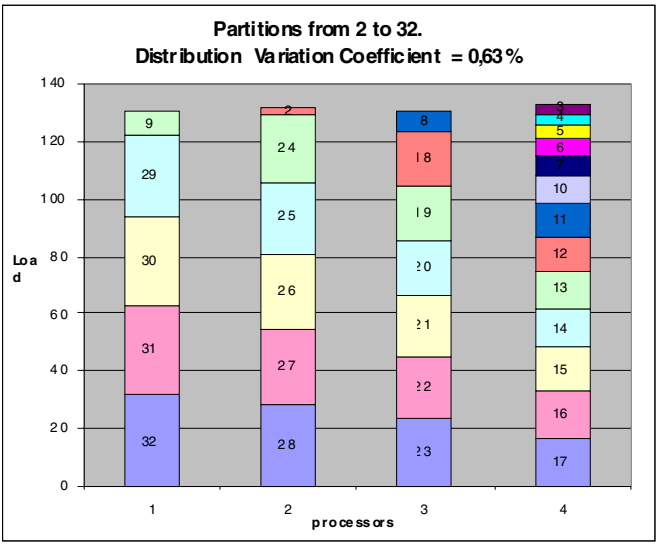


Fig. 1. Knapsacks generated for partitions range of 2 to 32 partitions distributed among four processors

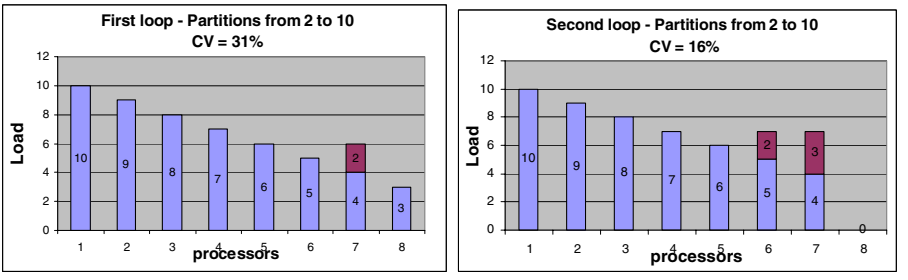


Fig. 2. Knapsacks generated in the first two iterations of the processing of the load balance algorithm when input variation coefficient = 20%

5 Results and Discussion

5.1 Environment and Test Description

The SGI ALTIX 450 Venus machine with 32 Intel Itanium2 processor cores (1.6 GHz) e 64 GB of memory from the High Performance Computing Center (NACAD) of COPPE/UFRJ was used for execution and performance analysis of this work. Jobs execution was controlled by PBS (Portable Batch System) job scheduler. The application was developed using the C programming language and the Message Passing Interface (MPI) for processors communication.

The tests were conducted with a synthetic file of one million record and seven variables. The file was processed for ranges of 3, 7, 15 e 31 partitions. The range of three partitions had 2, 3 and 4 clusters, the range of seven partitions had values from 2 up to 8 clusters, the range of fifteen partitions had values from 2 up to 16 clusters and the range of 31 partitions had values from 2 up to 32 clusters.

5.2 Results and Analysis

The load balanced FCM cluster analysis algorithm shows a significant reduction in processing time when compared to the prior approach [3] as showed in Table 1.

Table 1. Processing times of the synthetic dataset

Processors	Partitions Range / Time (seconds)			
Balaced Algorithm	2 - 4	2 - 8	2 - 16	2 - 32
1	62.576763	238.519065	957.624869	4,042.916829
4	27.659887	65.000000	246.093375	1,026.675303
8	13.440102	46.530077	132.000000	517.987720
12	9.103440	31.832853	90.000000	352.376751
Unbalanced Algorithm				
1	119.448050	475.618360	1,794.868058	7,350.881991
4	30.383583	114.260539	451.347466	1,837.258185
8	15.636881	57.642829	224.130619	909.555979
12	10.358894	38.253603	148.851859	616.004967

Table 2. Ratio between partitions number and processors numbers

Ratio Partitions / Processors				
Procs	Number of Partitions			
	3	7	15	31
1	3.00	7.00	15.00	31.00
4	0.75	1.75	3.75	7.75
8	0.38	0.88	1.88	3.88
12	0.25	0.58	1.25	2.58

The best processing times happen when the rate of number of partitions by number of processors has the highest values (Table 2), which means that communication is the biggest hindrance to the good performance of the algorithm. On the other hand, when the values rate go lower, the time reduction rate decreases.

The balanced algorithm reduced the processing time in all tests (Table 3), but the best time savings advantages are for the biggest rates of number of partitions by number of processors.

The unbalanced algorithm speed up and efficiency values where compared to the balanced algorithm single processor time because the sequential processing time of this algorithm was the best of them.

The load balancing algorithm scales well for an increasing number of processors presenting good speed up and efficiency values, and in all cases presents better speed up and efficiency values than the unbalanced algorithm (Fig. 4) (Fig. 5).

The balanced algorithm presents best speed up values when processing bigger range of partitions revealing that minimizing communications is a very effective way of reducing parallel processing time.

Table 3. Percentage reduction values from comparing balanced and unbalanced algorithms when processing the one million lines dataset

	Partition´s Range			
Processors	4	8	16	32
1	47.61%	49.85%	46.65%	45.00%
4	8.96%	43.11%	45.48%	44.12%
8	14.05%	19.28%	41.11%	43.05%
12	12.12%	16.78%	39.54%	42.80%

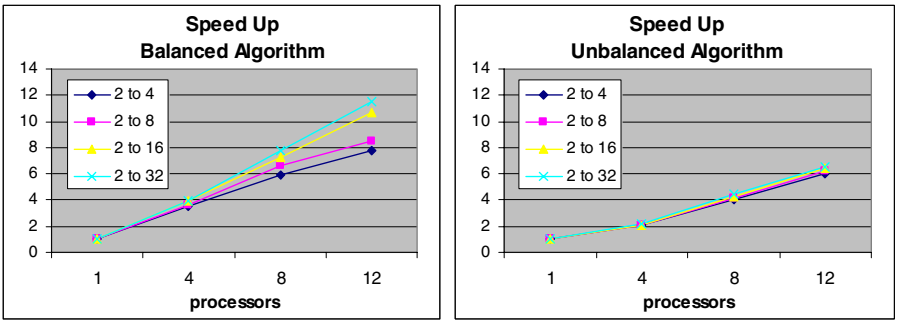
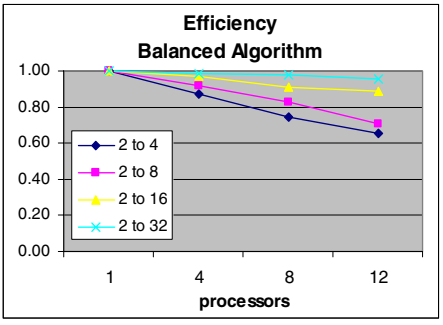


Fig. 3 & Fig. 4. Balanced and unbalanced algorithms speed up values for one million lines dataset processing for different ranges of partitions



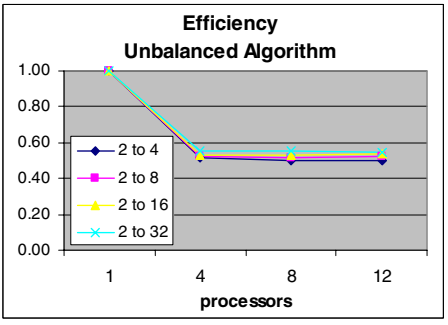


Fig. 5 & Fig. 6. Balanced and unbalanced algorithms efficiency values for one million lines dataset processing for different ranges of partitions

6 Conclusions

This work presents a significant improvement to the performance of the parallel FCM cluster analysis algorithm [3]. The load balancing for FCM cluster analysis algorithm scales well and presents good efficiency for all parallel contexts, bringing new levels of performance to the parallel FCM cluster analysis.

Nevertheless, the applicability of this approach has to be improved with a strategy for establishing a lower bound for the algorithm, in order to not keep assigning processors to evaluate charges when there is no benefit from parallel execution, as in the situations of the analysis of small range of partitions. In such cases the parallel proposed approach would keep using machine capacity without any benefit.

Acknowledgements

This work has been supported by the Brazilian Research Council (CNPq), by the Brazilian Innovation Agency (FINEP) and by the National Petroleum Agency (ANP). The authors are grateful to High Performance Computing Center (NACAD-COPPE/UFRJ) where the experiments were performed.

References

[1] Boutsinas, B., Gnardellis, T.: On distributing the clustering process. Pattern Recognition Letters 23, 999–1008 (2002)

[2] Rahimi, S., Zargham, M., Thakre, A., Chhillar, D.: A parallel Fuzzy C-Mean algorithm for image segmentation. In: Proceedings of the IEEE Annual Meeting of the Fuzzy Information NAFIPS 2004, vol. 1, pp. 234–237 (2004)

[3] Modenesi, M.V., Costa, M.A., Evsukoff, A.G., Ebecken, N.F.F.: Parallel Fuzzy c-Means Cluster Analysis. In: Daydé, M., Palma, J.M.L.M., Coutinho, Á.L.G.A., Pacitti, E., Lopes, J.C. (eds.) VECPAR 2006. LNCS, vol. 4395, pp. 139–142. Springer, Heidelberg (2007)

[4] Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Computing Surveys 31(3), 264–323 (1999)

[5] Bezdek, J.C.: Pattern Recognition with Fuzzy Objective Function Algorithms. Plenum, New York (1981)
[6] Xie, X.L., Beni, G.A.: Validity measure for fuzzy clustering. IEEE Transactions on Pattern Analysis and Machine Intelligence 3(8), 841–846 (1991)
[7] Bezdek, J., Pal, N.R.: Some new indexes of cluster validity. IEEE Trans. Systems Man and Cybernetics B 28, 301–315 (1998)
[8] Pakhira, M.K., Bandyopadhyay, S., Maulik, U.: Validity index for crisp and fuzzy clusters. Pattern Recognition 37, 487–501 (2004)
[9] Dantzig, G.B.: Discrete Variable Extremum Problems. Operations Research 5, 266–277 (1957)
[10] Mitten, L.G.: Branch-And-Bound Methods: General Formulation and Properties. Operations Research 18(1), 24–34 (1970)
[11] Martello, S., Toth, P.: Knapsack Problems, Algorithms and Computer Implementations. John Wiley & Sons, Chichester (1990)
[12] Pissinger, D., Toth, P.: Knapsak problems. In: Du, D.-Z., Pardalos, P. (eds.) Handbook of Combinatorial Optimization, vol. 1. Kluwer Academic Publishers, Dordrecht (1998)
[13] Johnson, D.: Near-Optimal Bin Packing Algorithms, Doctoral Thesis. MIT, Cambridge (1973)
[14] Graham, R.L.: Bounds on multiprocessor timing anomalies. SIAM J. Appl. Math. 17, 416–429 (1966)

Scalable Parallel 3d FFTs for Electronic Structure Codes

Andrew Canning

CRD, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA
Department of Applied Science, University of California, Davis, CA 95616, USA

Abstract. First-principles methods based on Density Functional Theory (DFT) where the wavefunctions are expanded in plane waves (Fourier components) are the most widely used approach for electronic structure calculations in materials science. The scaling of this method depends critically on having an efficient parallel 3d FFT that minimizes communications and calculations. We present an implementation and performance data of a parallel 3d FFT specifically designed for electronic structure calculations that scales to thousands of processors on leading parallel and vector computer platforms (IBM SP, Cray XT, NEC SX).

1 Parallel Implementation of 3d FFT

First-principles methods based on Density Functional Theory (DFT) in the Kohn-Sham (KS) [1] formalism are the most widely used approach for electronic structure calculations in materials science. The most common implementation of this approach involves the expansion of the wave functions in plane waves (Fourier components) and the use of pseudopotentials to replace the nucleus and core electrons. In this implementation we require parallel 3d FFTs to transform the electronic wavefunctions from Fourier space to real space to construct the charge density. Parallel 3d FFTs are also required in other parts of the code e.g. to transform potential terms from real space to Fourier space. This gives a computationally very efficient approach with a full quantum mechanical treatment for the valence electrons, allowing the study of systems containing hundreds of atoms on modest-sized parallel computers. Taken as a method DFT-based codes are one of the largest consumers of scientific computer cycles around the world with theoretical chemists, biologists, experimentalists etc. now becoming users of this approach. Parallel 3d FFTs are very demanding on the communication network of parallel computers as they require global transpositions of the FFT grid across the machine. The ratio of calculations to communications for 3d FFTs is of order $\log N$ where N is the grid dimension (compared to a ration of N for a distributed matrix multiply of matrix size N) which makes it one of the most demanding algorithms to scale on a parallel machine. A scalable parallel 3d FFT is critical to the overall scaling of plane wave DFT codes.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 280–286, 2008.

The Kohn-Sham formalism of DFT within the Local Density Approximation (LDA) requires that the wavefunctions of the electrons $\{\psi_i\}$ satisfy

$$[-\frac{1}{2}\nabla^2 + \sum_R v_{ion}(r-R) + \int \frac{\rho(r')}{|r-r'|} d^3r' + \mu_{xc}(\rho(r))]\psi_i = \varepsilon_i \psi_i \qquad (1)$$

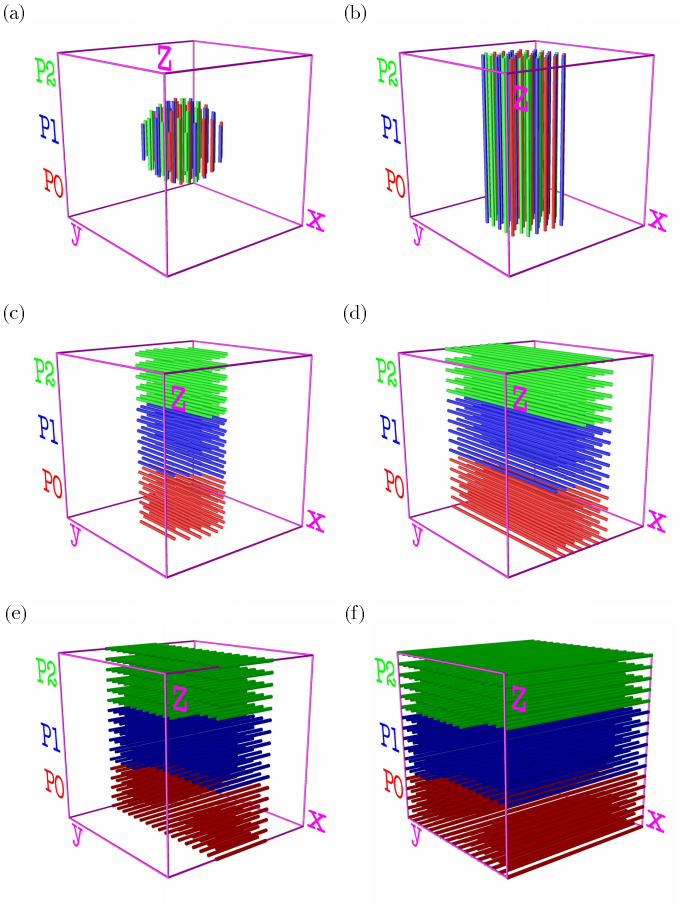
where $v_{ion}(r)$ is the ionic pseudopotential, $\rho(r)$ is the charge density and $\mu_{xc}(\rho(r))$ is the LDA exchange-correlation potential. We use periodic boundary conditions, expanding the wavefunctions in plane waves (Fourier components),

$$\psi_{j,\mathbf{k}}(\mathbf{r}) = \sum_{\mathbf{g}} a_{j,\mathbf{k}}(\mathbf{g}) e^{i(\mathbf{g}+\mathbf{k}).\mathbf{r}} \; . \qquad (2)$$

The selection of the number of plane waves is determined by a cutoff E_{cut} in the plane-wave kinetic energy $\frac{1}{2}|\mathbf{g}+\mathbf{k}|^2$ where $\{\mathbf{g}\}$ are reciprocal lattice vectors. This means that the representation of the wavefunctions in Fourier space is a sphere or ellipsoid with each $\mathbf{g}$ vector corresponding to a Fourier component (see figure 1(a)). The $\mathbf{k}$'s are vectors sampling the first Brillouin Zone (BZ) of the chosen unit cell (or supercell). The Kohn-Sham equations are usually solved by minimizing the total energy with an iterative scheme, such as conjugate gradient (CG), for a fixed charge density and then updating the charge density until self-consistency is achieved (for a review of this approach see reference [2]). Some parts of the calculation are done in Fourier space and some in real space transforming between the two using 3d FFTs. Our particular implementation in PARATEC (PARAllel Total Energy Code) [3] is based on a Grassmann conjugate gradient minimization [4] where all bands are minimized simultaneously. This allows us to use efficient BLAS3 routines for many parts of the calculation and also to block the communications to ensure MPI messaging is not latency dominated in the 3d FFTs.

The two most important criteria driving the choice of any parallelization strategy are equal division of the computational workload among the processors (load balancing) and minimization of the communications. We distribute the $\mathbf{g}$ vectors for each band among the processors by giving out columns of $\mathbf{g}$ vectors to each processor (see figure 1(a)). These columns are of different length depending on where they are in the sphere with the longest columns cutting the center of the sphere. The computations in PARATEC that are performed in Fourier space (e.g. non-local pseudopotential and orthogonalization) are load balanced by assigning each processor approximately the same number of $\mathbf{g}$ vectors. The load-balancing algorithm first orders the columns in descending order, and then distributes them among the processors such that the next-available column is assigned to the processor containing the fewest $\mathbf{g}$ vectors. The number of $\mathbf{g}$ vectors a processor has corresponds to the total length of columns it holds. It is necessary to distribute complete columns of $\mathbf{g}$ vectors to each processor as the first step in the 3d FFT performs 1d FFTs on columns of $\mathbf{g}$ vectors. The real-space data layout of the wavefunctions is on a standard Cartesian grid, where each processor holds a contiguous part of the space arranged in columns, as shown in figure 1(f). Each processor holds the same number of columns (to within one

FIGURES



1

Fig. 1. Parallel three dimensional FFT. This figure shows which processors deal with which part of the grid during the three dimensional FFT. The colors red, blue and green correspond to the part of the grid that resides on processors zero to two.

column) which load balances the real space part of the calculation. The charge density is constructed by performing 3d FFTs on each wavefunction to obtain the wavefunction on the larger real space grid. The wavefunctions are then squared and summed on this grid to produced the charge density that is then used in the calculation of the potential for the next self-consistent step in the solution of the Kohn-Sham equations.

A 3d FFT consists of three sets of 1d FFTs in the x,y and z directions with transpositions of the data between each set of 1d FFTs. Only two transposes are needed if the final data layout is not required to have the same x,y,z order in both spaces. Since the **g** vectors are distributed across the processors these two transposes can require global communications across the parallel computer and are the most communication intensive part of the whole calculation. We have therefore written a specialized 3d FFT to minimize the amount of communications. This 3d FFT is different from a standard 3d FFT as we have a sphere of points in Fourier space rather than a standard grid. This 3d FFT takes advantage of the fact that the real space grid is usually about twice the diameter of the sphere and at each of the three sets of 1d FFTs this sphere is in a sense expanding into the larger grid. We therefore only perform 1d FFTs and communications on the non-zero data elements which greatly reduces the amount of communications compared to using a standard library routine for the 3d FFT. Also when performing the second transpose to the final real space data layout (see figure 1) we choose the data layout to have as closely as possible complete planes of data on each processor so that the transpose is local and there is little data communication. In this way it is only the first transpose on the smaller data set where there is significant communication. Our 3d FFT can run on any number of processors for any grid and sphere size. If we used vendor supplied 3d FFTs we would have restrictions on grid sizes as well as performing more calculations and communications than our specialized 3d FFT since the grid size in Fourier and real space would have to be the same. Our specialized 3d FFT is numerically equivalent to using a vendor supplied 3d FFT. The details of each step in our Fourier space to real space 3d FFT are (with z,y,x ordering in Fourier space and x,y,z in real space):

1. Each processor pads out the ends of each of the z-columns of g vector coefficients that it holds with zeros to form full length z-columns on each processor. The complete data set is now a cylinder of length $2d$ and diameter d where d is the diameter of the original g vector sphere and $2d$ is the cube size (see figure 1(b)).
2. Each processor performs one-dimensional FFTs on its set of z-columns.
3. The cylinder of data is now reorganized from z-columns to y-columns (ordered by their x,z indices) with each processor now holding a contiguous set of y-columns. Global data redistribution is required at this step (ie. going from figure 1(b) to figure 1(c)),as can be seen by the changes in color of the data elements. Each processor is given as closely as possible the same number of y-columns.
4. The y-columns (which are sections through the cylinder) are now padded with zeros at the ends to form full length columns. The complete data set is now a slab of dimension d in the x direction and $2d$ in the other directions (see figure 1(d)).
5. Each processor performs one-dimensional FFTs on its set of y-columns.
6. The slab of data is now transformed from y-columns (x,z ordered) to x-columns (y,z ordered) with each processor now having a set of contiguous

x-columns (ie. going from figure 1(d) to figure 1(e)). Each processor is given as closely as possible the same number of x-columns. Communications are minimized at this step since most of the transformations are local to the processor with only data at the interfaces of the colored blocks being communicated. In the ideal case where there are complete (y, x) planes on each processor the transpose can be done locally on each processor and there are no communications. Due to our choice of data layouts in the FFT the main communications are in step 3 where the data set (the cylinder) is much smaller than the slab.

7. The x-columns are now padded at the ends with zeros so the global data set is now the complete cube of side $2d$ (see figure 1(f)).
8. Each processor performs one-dimensional FFTs on its set of x-columns producing the final distributed real space representation of the wavefunction in x,y,z order.

An inverse 3d FFT is the reverse of these steps. While it is important to minimize the amount of data transfer in 3d FFTs, communication latency can also become a major issue. In the first transpose in the 3d FFT all the processors are sending data to all the other processors so the data packet size (for a fixed size physical system) scales as the inverse of the number of processors squared. Therefore as we scale up to thousands of processors the data packets can become very small and communication latencies can dominate the code. To avoid this problem in our code we use an all-band method that allows us to perform many 3d FFTs at the same time and block the communications. There is an input parameter in our code which chooses the number of 3d FFTs to be performed at the same time. In this way, at the cost of using more memory, we can increase the packet size of the communications in the 3d FFTs to avoid the latency problem. For machines with higher latency like the IBM SP we have found that this can increase the speed of the code by 50-100% on runs in the hundreds of processor regime. For large processor counts we typically do up to fifty 3d FFTs at the same time which greatly reduces the latency problem.

2 Code Details and Performance

PARATEC is written in F90 and MPI and is designed primarily for massively parallel computing platforms, but can also run on serial machines. The code has run on many computer architectures and uses preprocessing to include machine specific routines such as the one dimensional FFT calls which are used in our specialized 3d FFTs. For the parallel vector platforms (Cray X1 and NEC SX) an efficient vector implementation of the one dimensional FFT libraries was required. The standard vendor supplied 1D FFT routines (on which our own specialized 3D FFTs are written) run at a relatively low percentage of peak. Code transformation was therefore required to rewrite our 3D FFT routines to use simultaneous (often called multiple) 1D FFT calls, which allow effective vectorization across many 1D FFTs. Additionally, compiler directives were inserted

Table 1. PARATEC results for a 488 atom CdSe quantum dot on the different platforms. The real space grid size for the 3d FFTs is 252*3. Bassi is an IBM SP with eight Power 5 processors per node, located at the NERSC computer center, Lawrence Berkeley National Laboratory. Th under is an Intel Itanium2 cluster with four processors per node and a Quadrics interconnect, located at Lawrence Livermore National Laboratory. Phoenix is a Cray X1E vector architecture located at Oak Ridge National Laboratory. The ES is the Earth Simulator which is a custom designed NEC SX6 located at the Earth Simulator Center, Yokohama. Franklin is a Cray XT4 with dual core Opteron processors and a 3d Torus interconnect, located at the NERSC computer center, Lawrence Berkeley National Laboratory.

	Bassi		Thunder		Phoenix		ES		Franklin	
P	IBM SP P5		Itanium2		Cray X1E		NEC SX6		Cray XT4	
	Gflop/P	%Pk	Gflop/P	%Pk	Gflop/P	%Pk	Gflop/P	%Pk	Gflop/P	%Pk
64	—	—	—	—	4.88	27	—	—	—	—
128	5.49	72	2.84	51	3.80	21	5.12	64	—	—
256	5.52	73	2.63	47	3.24	18	4.97	62	3.36	65
512	5.13	67	2.44	44	2.22	12	4.36	55	3.15	61
1024	—	—	1.77	32	—	—	3.64	46	2.93	56
2048	—	—	—	—	—	—	2.67	33	2.65	46

to force the vectorization and multistreaming (on the X1) for loops that contained indirect addressing. The main communications in the code are performed in the parallel 3d FFTs with most of the other parts of the code performing dense linear algebra on their local data. For the data presented in this paper the 3d FFTs typically take about 30% of the total runtime. The other parts of the code run at a high percentage of peak as they are mainly performing dense linear algebra on large matrices. Table 1 presents performance data for 3 CG steps of a 488 atom CdSe (Cadmium Selenide) quantum dot and a standard Local Density Approximation (LDA) run of PARATEC with a 35 Ry cut-off using norm-conserving pseudopotentials. The real space grid size for the 3d FFTs is 252 cubed and the calculation is for 709 bands. A typical calculation would require at least 60 CG iterations to converge the charge density for a CdSe dot. CdSe quantum dots are luminescent in the optical range at different frequencies depending on their size and can be used as electronic dye tags by attaching them to organic molecules. They represent a nanosystem with important technological applications.

As can be seen from Table 1 PARATEC obtains a high percentage of peak on both superscalar and vector based architectures. The machines with the best communication networks and lowest latency, such as the Cray XT4, have the best scaling to large processor counts for the 3d FFT and hence the whole code. The Power 5 chip has the highest per processor performance for this code. This code makes heavy use of Cache in the FFTs as well as the other dense linear algebra operations so that RISC type architectures obtain a percentage of peak that is similar to vector machines. The ES achieved the highest peak performance of 5.5 Tflops on 2048 processors with the Cray XT4 being only a few percent

slower. The Cray X1E obtained a lower percentage of peak due in part to some non-vectorizable sections of the code that run on the slow scalar processor. The NEC ES has a relatively faster scalar processor.

3 Discussion and Conclusions

In this paper we have present an efficient implementation of a parallel 3d FFT specifically designed for plane wave electronic structure codes. We have shown that with this 3d FFT our electronic structure code can scale well to thousands of processors on a variety of different computer architectures ranging from vector to superscalar. The limiting factor to scaling to larger processor counts is the communications in the 3d FFTs and we are investigating different communication schemes, such as using more collective operations, to allow us to scale to larger processor counts.

Acknowledgments

The author would like to thank L. Oliker J. Shalf and J. Carter for useful discussions and help in gaining access to different computer platforms. The author would also like to thank the staff of the Earth Simulator Center, especially Dr. T. Sato. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098. This research used resources of the Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

References

1. Kohn, W., Sham, L.J.: Phys. Rev. 140, A1133 (1965)
2. Payne, M., Teter, M.P., Allan, D.C., Arias, T.A., Joannopoulos, J.D.: Rev. Mod. Phys. 64, 1045 (1992)
3. Pfrommer, B., Raczkowski, D., Canning, A., Louie, S.G.: PARATEC (PARAllel Total Energy Code). Lawrence Berkeley National Laboratory, `www.nersc.gov/projects/paratec/` (with contributions from F. Mauri, M. Côté, Y. Yoon, C. Pickard and P. Haynes)
4. Raczkowski, D., Fong, C.Y., Schultz, P.A., Lippert, R.A., STechel, E.B.: Phys. Rev. B 64, 155203 (2001)

Evaluation of Sparse LU Factorization and Triangular Solution on Multicore Platforms*

Xiaoye S. Li

Lawrence Berkeley National Laboratory, MS 50F-1650,
One Cyclotron Road, Berkeley, CA 94720, USA
Tel: (510) 486-6684. Fax: (510) 486-5812
xsli@lbl.gov

Abstract. The Chip Multiprocessor (CMP) will be the basic building block for computer systems ranging from laptops to supercomputers. New software developments at all levels are needed to fully utilize these systems. In this work, we evaluate performance of different high-performance sparse LU factorization and triangular solution algorithms on several representative multicore machines. We include both pthreads and MPI implementations in this study, and found that the pthreads implementation consistently delivers good performance and a left-looking algorithm is usually superior.

1 Introduction

The Chip Multiprocessor (CMP) systems will be the basic building blocks for computers ranging from laptops to supercomputers. Compared to the superscalar microprocessors exploiting high degree of instruction level parallelism, the CMP designs represent a paradigm shift that strikes better trade-offs between performance (parallelism) and energy efficiency. In the case of multicore architecture, high performance is achieved by replicating the execution units on a single die while keeping the clock rate (hence power consumption) relatively low. In the case of multithreaded architecture, high throughput is achieved by providing multiple sets of hardware thread contexts for each FPU and simultaneously executing multiple streams of instructions without relying on speculation. Multithreading can effectively hide instruction and cache latency. In theory, the CMPs can often be programmed the same way as the conventional SMPs, but the CMPs have lower memory bandwidth and an abundance of fine-grained parallelism. Given the diversity of CMP designs, it is necessary, albeit difficult, to develop new software strategies at the system level as well as the application level in order to fully utilize the hardware resources.

In this paper, we study several kernel algorithms associated with sparse direct solvers on a couple of leading CMP systems. Direct solvers based on matrix

* This research was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 287–300, 2008.

factorizations are among the most reliable methods or preconditioners for solving sparse linear and eigen systems. They are often the computational bottlenecks in large-scale computer modeling codes. Over the past decade or so, we have been developing new algorithms to exploit advanced high-performance, large-scale parallel computers. Our algorithm research has led to the software package called SuperLU [4,7], which is widely used in research and industry.

Previously, Williams et al. [14] performed extensive study and optimization of sparse matrix-vector multiply (SpMV) on several leading CMP systems. In SpMV, the matrix A needs to be read only once, hence the ratio of flops to memory accesses is $\mathcal{O}(1)$. The operation is largely memory-bound, since there is hardly any reuse, They develoepd various blocking strategies exploiting different hardware components, including thread blocking, register blocking, cache and local store blocking, and TLB blocking. In contrast, the factorization algorithm need to access the matrix A, the L and U factors multiple times, which exhibit higher reuse. But the ratio of flops to memory accesses changes throughout the elimination process. In early stages of elimination, the factors are sparser and workload is memory-bound. Whereas in later stages, the factors are denser; level 3 BLAS are appropriate and hence the workload is compute-bound. The change of arithmetic density makes it harder to develop uniform strategies for performance optimization.

The goal of this study is two-fold. Firstly, we would like to evaluate performance of the existing implementations on the new CMP architectures, and secondly, we would like to identify the inefficiencies in the algorithms and/or implementations and determine ways to improve them for the new architectures.

2 Experimental Machines

Our testing systems include an Intel Colvertown, a Sun VictoriaFalls, and an IBM Power5. The last one contains a conventional SMP node.

The Intel Colvertown consists of two sockets, each with two pairs of dual-core Xeon chips (Core2Duo), with total eight processors (Dell PowerEdge 1950 dual-socket). Each core runs at 2.33 GHz with a peak performance of 9.3 Gflops (4 flops per cycle), and has a private 32 KB L1 cache. Each chip (two cores) share a 4 MB L2 cache. Each socket has access to a Front Side Bus (FSB) delivering 10.6 GB/s. The two independent FSBs are connected to the memory controller which interfaces to the DRAM channels, delivering 21.3 GB/s read memory bandwidth and 10.6 GB/s write bandwidth.

The dual-chip Sun VictoriaFalls contains 16 SPARCv9 cores, in which each CMP is a Niagara2 chip with 8 cores. Each core runs at 1.16 GHz with a peak performance of 1.16 Gflops, and has a private 8 KB L1 cache. All eight cores share a 4 MB L2 cache. In addition, each core supports eight hardware threads, and the entire dual-chip system provides a total of 128 threads. The two sockets are interconnected via External Coherence Hubs (ECH). There are altogether 8 FBDIMM memory channels, delivering the aggregate DRAM bandwidth of 42.6 GB/s for read and 21.3 GB/s for write.

The IBM p575 Power5 is a scalable distributed memory HPC system consisting of conventional SMP nodes. The entire system (bassi at NERSC) has 111 compute nodes, each of which has 8 Power5 processors running at 1.9 GHz and has a shared memory pool of 32 GBytes. Each processor has a peak performance of 7.6 GFlops (4 flops per cycle), and has a private 32 KB L1 cache. We only use one SMP node in this study.

Table 1 summarizes the key architectural features of the three systems used in this study. Figure 1 shows the simplified block diagrams of the two CMP systems. The sources come from [11,12,14].

Several characteristics in Table 1 are worth noting. Compared to the IBM SMP node, the cores-to-DRAM bandwidths are considerably lower on the multicore systems. In addition, the write bandwidth on both multicore systems is only half of the read bandwidth. The byte-per-flop ratio shows the balance of the memory bandwidth versus floating-point speed, with larger ratio indicating higher bandwidth relative to core speed. The CMP systems are clearly worse than conventional SMPs in this regard, implying that algorithms demanding larger memory bandwidth will be penalized in performance. The conventional SMPs usually have more complex designs and consume more power, see the last line in the table. The quoted number for p575 was measured while running a full computational workload, which is much lower than the manufacturer's peak power rating [12].

Table 1. Summary of the experimental machines

Systems	Intel Colvertown	Sun VictoriaFalls	IBM Power5 (575)
Core type	superscalar (4)	multithreaded (8)	superscalar (4)
Clock (GHz)	2.3	1.16	1.9
L1 Dcache	32 KB	8 KB	32 KB
DP Gflops	9.3	1.16	7.6
# Sockets	2	2	8
# cores/socket	4	8	1
L2 cache	4 MB/2-cores (16 MB)	4 MB/socket (8 MB)	1.92 MB /core (32 MB L3$/node)
DP Gflops	74.7	18.7	60.8
DRAM GB/s read	21.3	42.6	200
write	10.6	21.3	–
Byte/flop ratio	0.29	0.44	3.29
Power/socket (Watts)	160 (max)	84 (max)	500 (measured [12])

3 Overview of the Algorithms and Implementations

Over the years, we have been deveoping the SuperLU suite of libraries, with different variants of sparse Gaussian elimination targeted for shared or distributed memory high performance machines [7]. Below, we briefly summarize the algorithmic and implementational features of the two variants used in this study.

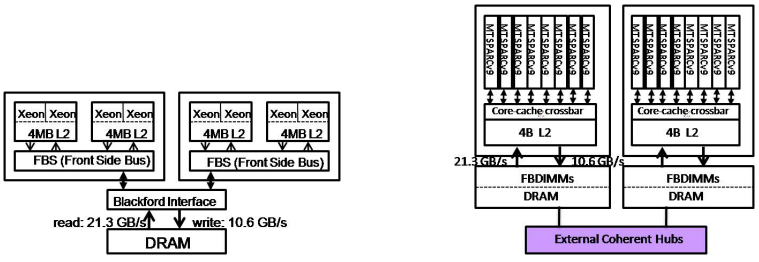


Fig. 1. High level block diagrams of Intel Clovertown (left) and Sun VictoriaFalls (right)

3.1 Factorization in SuperLU_MT Using Pthreads or OpenMP

The initial target platforms of SuperLU_MT were SMPs of modest size (e.g., 32 processors). It was first developed with Pthreads, and recently, we have added OpenMP support. The earlier tests on a number of commercially popular SMPs, such as Sun, DEC Alpha, SGI Origin, and Cray C90/J90, demonstrated excellent speedups [3,6]. Pleasantly, we will show that this code is equally suitable for the modern multicore machines. The algorithmic features and parallelization techniques are outlined below:

- Use panel-based *left-looking* factorization, with partial pivoting and possibly with diagonal preference to better preserve sparsity. Use supernode-panel update kernel to effectively use Level 3 BLAS.
- Use an asychronous and barrier-free dynamic algorithm to schedule both coarse-grain and fine-grain parallel tasks to achieve a high level of concurrency. A globally shared task queue is used to store the ready panels in the column elimination tree, and whenever a thread becomes free, it obtains a ready panel from the task queue. The coarse-grain task is to factorize the independent panels in the disjoint subtrees, while the fine-grain task is to update panels by previously computed supernodes. The scheduler facilitates the smooth transition between the two types of tasks, and maintains load balance dynamically.

Figure 2 illustrates the left-looking factorization scheme, and the dynamic scheduling method using the elimination tree.

3.2 Factorization in SuperLU_DIST Using MPI

The target platforms of this code are the massively parallel distributed memory computers [8]. Previously, we tested this code on a number of HPC platforms, including Cray T3E, IBM SP, and various Linux clusters. Good scalability was demonstrated up to one thousand processors. In order to address the scalability issues, the parallel algorithm is significantly different from that in SuperLU_MT. The main differences are in pivoting strategy and matrix distribution, which are summarized below.

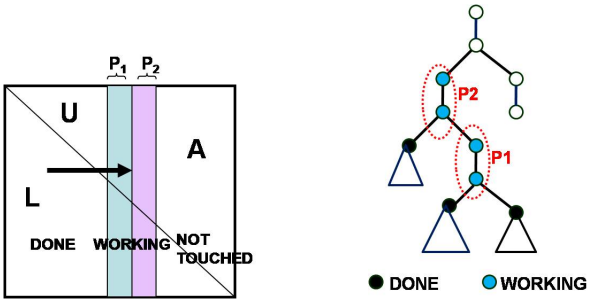


Fig. 2. Panel-based left-looking algorithm in superlumt

– Use block-based *right-looking* factorization, which exhibits a high degree of parallelism during the block outer-product updates to the trailing submatrices. According to the supernode partition, perform a two-dimensional (nonuniform) block-cyclic matrix-to-processor mapping. Use the elimination DAGs to identify task and block dependencies, and a look-ahead mechanism to better overlap communication with computation and shorten the critical path.
– Before factorization, pre-permute the rows of the matrix so that the diagonal has entries of large magnitude, using a weighted bipartite matching algorithm from MC64 [5]. During factorization, allow single precision perturbation to the small diagonal entries.

Figure 3 illustrates the 2D block-cyclic partition and distribution for a sparse matrix.

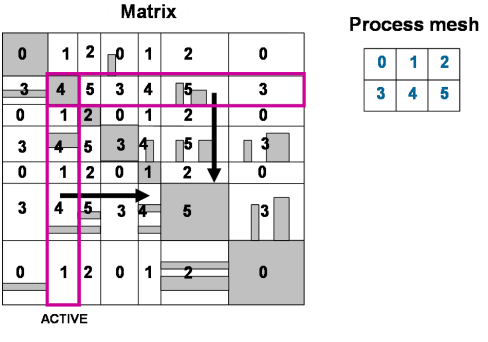


Fig. 3. Block-based right-looking algorithm in `SuperLU_DIST`

3.3 A Note on `SuperLU_MT` — Symmetric Mode

In order to conduct direct comparision between `SuperLU_MT` and `SuperLU_DIST`, we have added a new algorithmic choice for `SuperLU_MT`, which is called *symmetric mode*. As is known, with partial pivoting (`SuperLU_MT`), it is better to use

A^TA-based column ordering strategy to preserve sparsity, since pattern-wise, the Cholesky factor of A^TA upper bounds the LU factors in the decomposition $PA = LU$, for any row permutation P. However, in the case of satic pivoting where pivots are selected before hand (`SuperLU_DIST`), no row interchanges are made during factorization, then the A^TA-based upper bound becomes too loose. Therefore, we can use a tighter upper bound based on $A + A^T$. The difference in the amount of fill using an A^TA-based sparsity ordering or an $(A + A^T)$-based one can be more than a factor of two. The new symmetric mode option in `SuperLU_MT` contains an algorithm that is similar to the one in `SuperLU_DIST` — it uses MC64 to perform static numerical pivoting, an $(A+A^T)$-based symmetric sparsity ordering, and single precision diagonal perturbations when needed.

All our experimental results used the symmetric mode in `SuperLU_MT`. Thus, the amount of fill and number of floating-point operations are roughly the same with both solvers.

3.4 Triangular Solution in `SuperLU_DIST`

The triangular solution phase in `SuperLU_MT` is not yet parallel, therefore we will evaluate only the parallel algorithm in `SuperLU_DIST`. In $Lx = b$, where L is a lower triangular matrix, the i-th solution component is computed as

$$x_i = \frac{b_i - \sum_{j=1}^{i-1} L_{ij} \cdot x_j}{L_{ii}} \ .$$

Therefore, computation of x_i needs some or all of the previous solution components $x_j, j < i$, depending on the sparsity pattern of the i-th row of L. This sequentiality often poses a scaling hurdle for a parallel algorithm. Another hurdle to achieving good performance is the much lower arithmetic density as measured by flops per byte of DRAM access or communication, compared to factorization.

In the current implementation of `SuperLU_DIST`, the parallel triangular algorithm uses the same 2D block-cyclic distribution as used in the factorization phase. Figure 4 illustrate such a distribution and the solution process. The processes owning the diagonal blocks (called *diagonal processes*) are responsible for computing the corresponding blocks of the x components. When x_j is needed in $L_{ij} \cdot x_j$, and the owners of x_j and L_{ij} are different, x_j's processor needs to send it to the processor of L_{ij}, see ① in Figure 4. In case of ②, no communication is needed because both x_j and L_{ij} reside on the same processor, i.e. processor 1. After receiving the needed x_j entries, each processor proceeds with local summation, i.e., step ③ in Figure 4. Finally, the local sums are sent to the diagonal processor which performs the division, see ④ in the figure.

3.5 Entire Solvers

Since the numerical pivoting methods are different in the two solvers — partial pivoting in `SuperLU_MT` and static pivoting in `SuperLU_DIST`, the high level structure of the two codes are different. In case of partial pivoting, the fills are

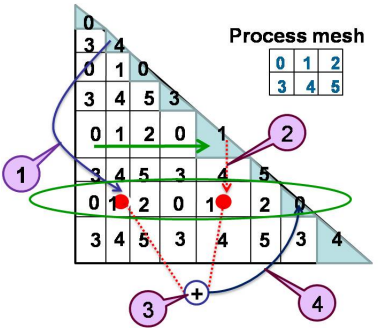


Fig. 4. Block-based triangular solution

generated dynamically, so the symbolic factorization step cannot be separated from numerical factorization. Whereas with static pivoting, we can separate symbolic and numerical factorization steps. Figure 5 summarizes the major steps of the two solvers and highlights the differences between them.

`SuperLU_MT`

1. Sparsity ordering
2. Factorization ... interleaveing:
   - 2.1) partial pivoting
   - 2.2) symbolic factorization
   - 2.3) numerical factorization (BLAS 2.5)
3. Triangular solution

`SuperLU_DIST`

1. Static pivoting
2. Sparsity ordering
3. Symbolic factorization
4. Numerical factorization (BLAS 3)
5. Triangular solution

Fig. 5. Major steps in the entire solvers.

4 Experimental Results

Table 2 presents the characteristics of our benchmarking matrices, which are available from the University of Florida Sparse Matrix Collection [2]. In the following subsections, we will present the parallel runtimes and analysis. We benchmarked the Pthreads version of `SuperLU_MT`, and `SuperLU_DIST` using MPICH [9].

4.1 Characterization from Hardware Performance Counters

Given that the memory system performance plays increasingly significant role on the CMP architectures and with the sparse matrix algorithms, it would be more relevant to quantify the memory access patterns of the different algorithms than to count the flops alone. For simpler kernels like SpMV, it is possible to count manually. For complex codes, we need to resort to performance analysis tools.

Table 2. Properties of the test matrices. Minimum degree algorithm was applied to the structure of $|A|+|A|^T$. "fill-ratio" denotes the ratio of number of nonzeros in $L+U$ over that in A; "Mean S-node" refers to an average number of columns in a supernode.

	application	dimension	nonzeros in A	fill-ratio	Mean S-node
g7jac200	economic model	59,310	837,936	40.2	1.9
stomach	duodenum model	213,360	3,021,648	45.4	4.0
torso1	2D model of torso	116,158	8,516,500	3.1	4.0
twotone	nonlinear anal. circuit	120,750	1,224,224	9.3	2.3

The first tool we used is PAPI [10] which provides an API to access machines' hardware counters. As a first cut, we examine the load and store instruction counts, which are independent of the cache memory organization. Table 3 compares the counts of the left-looking (`SuperLU_MT`) and right-looking (`SuperLU_DIST`) factorization algorithms. It is clear that right-looking algorithm incurs many more load and store instructions, typically an order of magnitude more.

Table 3. Factorization load and store instruction counts (billions), reported by PAPI

	LOAD		STORE	
	`SuperLU_MT`	`SuperLU_DIST`	`SuperLU_MT`	`SuperLU_DIST`
g7jac200	1.2	27.7	0.3	8.2
stomach	0.8	52.0	0.3	10.8
torso1	9.1	17.9	2.8	4.5
twotone	1.2	18.6	0.2	8.4

Although the load/store instruction count indicates the superiority of the left-looking algorithm in the sense of program's static behavior, we are also interested in the temporal behavior of the codes while running on an actual machine. Unfortunately, the two multicore machines do not yet have proper PAPI support for such study. So we used the CrayPat performance tool provided on the Cray XT systems [1]. CrayPat uses PAPI's counters to collect raw data, and then computes a variety of derived quantities which are easy to understand. The machine we used is the Cray XT4 installed at NERSC. Each node consists of a 2.6 GHz dual-core AMD Opteron processor, sharing 4 GBytes of memory. Each core has a 64KB L1 data cache of and a 1MB L2 cache. The L2 cache is a victim cache which holds only the cache lines evicted from L1, whereas most data loaded from memory go directly to L1. We used only one core to run the codes and collected data shown in Table 4, and the data are for the entire solvers. We report two metrics: "Mem-to-D1" measures the amount of data transferred between memory and L1 data cache, and "L2-to-Mem" measures the data traffic between L2 and memory. In both metrics, we see that `SuperLU_DIST` requires considerably larger amount of data transfer.

Lastly, we examine the flop-to-load (store) ratio in the triangular soution phase. We compare the metric for the two distinct stages of the solver, one

Table 4. The solvers' memory traffic (billion bytes) reported by CrayPat

	Mem-to-D1		L2-to-Mem	
	SuperLU_MT	SuperLU_DIST	SuperLU_MT	SuperLU_DIST
g7jac200	10.7	17.5	3.9	15.1
stomach	25.1	24.9	16.0	16.3
torso1	3.0	7.4	4.7	11.6
twotone	1.9	7.8	1.2	7.0

is "ordering + factor" and the other is "tri-solve". In Table 5, we report the respective ratios of flop-to-load and flop-to-store. As can be seen, in both metrics, the triangular solution phase has much smaller flop density, sometimes can be more than an order of mangnitude lower than the other part of the solver.

Table 5. Ratio of flops over load or store instructions in the triangular solution of SuperLU_DIST, compared with the rest of the program

	LOAD		STORE	
	ordering + factor	tri-solve	ordering + factor	tri-solve
g7jac200	0.86	0.14	2.89	0.24
stomach	1.35	0.24	6.49	0.47
torso1	0.75	0.21	2.95	0.35
twotone	0.30	0.06	0.67	0.09

4.2 Runtime

In the factorization codes of both solvers, the BLAS routines could take more than 30-40% of the time. Therefore the BLAS speed is a key performance bound. In sparse codes, the matrix size for BLAS calls is usually small. The kernel in SuperLU_MT is "BLAS 2.5", where we perform multiple DGEMV calls with different vectors while keeping the matrix in cache. Therefore, we usually keep the matrix size bounded by 200×100. The kernel in SuperLU_DIST is BLAS 3 (mostly DGEMM). In order to maintain good load balance, we use even smaller block sizes, such as 50×50. In Figure 6, we plot the performance (Gflops rate) of DGEMV and DGEMM on Clovertown and VictoriaFalls. In each case, we used the vendor's high performance mathematical libraries — Intel's MKL and Sun's SunPerf.

Recall that each core processor of VictoriaFalls is hardware-multithreaded. But without explicit parallelization (e.g., threading) at the software level, DGEMM only achieves less than one-third of the peak performance. That is, a single threaded program is incapable of fully utilizing the resources provided by a multithreading architecture.

Table 6 shows the parallel factorization times of the two solvers on Intel Clovertown. *For fair comparison, the time includes both symbolic and numerical factorization, because it is not possible to separate these two steps in* SuperLU_MT*, see Figure 5.*

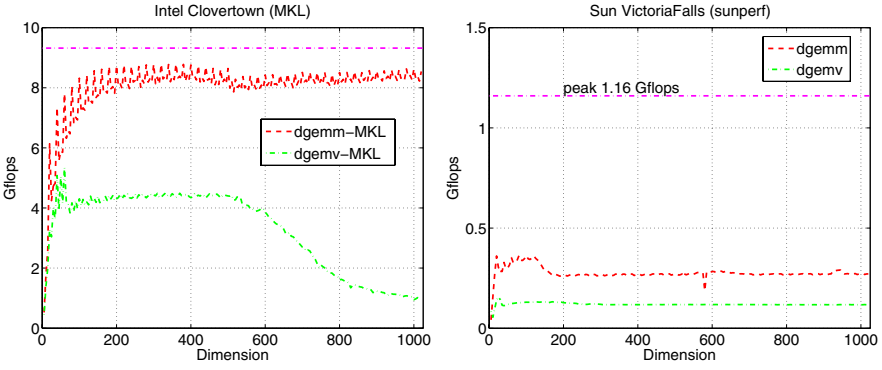


Fig. 6. BLAS performance on Intel Clovertown (left) and Sun VictoriaFalls (right). The top dashed line shows the core's peak performance.

First we note that MPICH can be configured with either `ch_shmem` device for shared memory processors, or `ch_p4` device for communication through sockets on distributed memory machines. We first used the default `ch_p4` configuration on the Clovertown cluster, and found that the code slowed down significantly beyond two or four cores. After we switched to `ch_shmem` setup, we obtained respectable speedup. Therefore, for a large distributed system comprising many-core chips, it is imperative to be able to configure MPICH in a *hybrid* device mode — `ch_shmem` mode within socket and `ch_p4` mode across sockets. Currently, the hybrid mode is not avaialbe.

Secondly, we examine the single core performance. We would expect that `SuperLU_DIST` outperforms `SuperLU_MT`, because the former uses BLAS 3 whereas the latter uses only BLAS 2.5. We see that this is true only with two matrices, g7jac200 and stomach, which have relatively denser L and U factors (the fill ratios are over 40, see Table 2), and hence BLAS 3 plays a larger role. For sparser problems, the algorithms are memory-bound. We believe the worse performance of `SuperLU_DIST` is mainly due to more memory traffic of the right-looking algorithm, especially more memory write operations, see the measures presented in Section 4.1.

Thirdly, we examine the speedups of the two codes. The last column of Table 6 shows the speedup obtained when creating eight threads or MPI tasks. The best speedup is 4.3 and is less than what we observed on conventional SMP processors [3]. After performing code profiling, we found that the overhead of the scheduling algorithm using the shared task queue and the synchronization cost using mutexes (locks) are quite small. Further study is needed to understand where the time goes.

In addition, `SuperLU_MT` usually achieves more speedup than `SuperLU_DIST`. This can be seen in the row "speedup ratio (`_MT/_DIST`)" associated with each matrix. In some cases, `SuperLU_MT` achieves a factor of two more speedup than `SuperLU_DIST`.

Table 6. Factorization time in seconds on Intel Clovertown

matrix	threads or tasks		1	2	4	8	speedup
g7jac200	SuperLU_MT		32.78	17.91	12.41	10.60	3.1
	SuperLU_DIST	ch_shmem	28.10	15.95	11.06	7.57	3.9
		ch_p4	28.62	22.98	56.31	62.39	
	speedup ratio (_MT/_DIST)		1.00	1.03	1.01	0.80	
stomach	SuperLU_MT		64.38	37.15	20.39	17.24	3.7
	SuperLU_DIST	ch_shmem	43.45	25.91	15.81	13.64	3.4
		ch_p4	44.28	27.84	210.99	264.58	
	speedup ratio (_MT/_DIST)		1.00	0.99	1.10	1.10	
torso1	SuperLU_MT		9.43	4.92	2.87	2.20	4.3
	SuperLU_DIST	ch_shmem	9.43	5.83	4.55	4.76	2.2
		ch_p4	9.62	7.23	54.77	76.32	
	speedup ratio (_MT/_DIST)		1.00	1.12	1.49	1.99	
twotone	SuperLU_MT		6.80	4.05	2.32	1.83	3.9
	SuperLU_DIST	ch_shmem	18.08	10.17	7.55	7.21	2.1
		ch_p4	18.34	12.19	47.30	60.99	
	speedup ratio (_MT/_DIST)		1.00	0.95	2.26	1.86	

Table 7 shows the parallel factorization times on the Sun VictoriaFalls. Recall that this system has 16 eight-way harware-threaded cores, and altogether we can have up to 128 threads. The single-thread performance of SuperLU_DIST is usually better than that with SuperLU_MT. This is probably because the machine has a higher byte-to-flop ratio (see Table 1) compared to Clovertown, hence it does not penalize an algorithm that is memory-bandwidth demanding, such as the right-looking algorithm in SuperLU_DIST.

However, the coarse-grain task parallelism supported by MPI programming does not match the fine-grain multithreading architecture — MPICH often crashes when more than 16 tasks are generated. The Pthreads programming is much more robust, and SuperLU_MT can effectively use 64 threads. Similar to the Clovertown results, SuperLU_MT usually achieves more speedup than SuperLU_DIST. This can be seen in the row "speedup ratio (_MT/_DIST)" associated with each matrix. In some cases, SuperLU_MT achieves a factor of 2 more speedup than SuperLU_DIST.

We see that SuperLU_MT achieves nearly perfect speedups for the first four to eight threads. This may be related to the Sun Solaris' round-robin scheduling policy which schedules multisocket first, then multicore, then multithreads [13]. With this order, the first few threads are spread across different sockets, and do not have much memory bus contention.

We now evaluate performance of the parallel triangular solution algorithm in SuperLU_DIST. We compare the eight-core Clovertown with the eight-processor Power5 SMP node. The parallel runtimes are tabulated in Table 8. The columns labeled "Current" correspond to the current implementation, and the columns labeled "Improved" refer to the new implementation as a result of this study.

It is very disapointing that on the Clovertown, the current code runs much more slowly with more cores involved. A similar trend was also observed on

Table 7. Factorization time in seconds on Sun VictoriaFalls. "f" indicates that an MPI failure occurred.

matrix	threads or tasks	1	2	4	8	16	32	64	128
g7jac200	SuperLU_MT	480.84	244.24	126.16	68.93	40.22	28.47	**23.95**	24.80
	SuperLU_DIST	283.44	153.18	83.09	49.20	31.70	f	f	f
	speedup ratio (_MT/_DIST)	1.00	1.06	1.09	1.15	1.24			
stomach	SuperLU_MT	1212.97	620.58	319.85	168.04	90.01	56.51	**53.54**	62.37
	SuperLU_DIST	598.49	329.28	183.90	116.22	85.56	f	f	f
	speedup ratio (_MT/_DIST)	1.00	1.06	1.13	1.33	1.79			
torso1	SuperLU_MT	201.05	102.09	52.51	27.41	15.16	11.56	**10.23**	11.34
	SuperLU_DIST	101.68	58.25	32.53	21.83	17.06	f	f	f
	speedup ratio (_MT/_DIST)	1.00	1.12	1.18	1.46	2.01			
twotone	SuperLU_MT	113.12	60.09	31.50	17.18	11.17	8.17	**7.26**	7.90
	SuperLU_DIST	135.43	78.44	46.64	30.01	18.49	f	f	f
	speedup ratio (_MT/_DIST)	1.00	1.08	1.19	1.38	1.26			

the VictoriaFalls. After we profiled various parts of the code, we found that the slowdown is due to many calls of `MPI_Reduce`; in fact, on eight cores, `MPI_Reduce` can take over 75% of the time. Consider one block row of the L matrix, as circled in Figure 4, the diagonal process 0 needs to know which off-diagonal processes (1 and 2) will have sum contributions to be sent to process 0. To compute this count, every process holds a 0/1 flag depending on whether this process has nonzero blocks. Then, all the processes in each process row perform an `MPI_Reduce` (by SUM) over the flags, with diagonal process being the root. Overall, each block row corresponds to one such reduction operation.

To address this issue, we have made the following improvement. Instead of performing many reductions with one integer, we allocate a flag array of integers, the size of which is the number of block rows owned by each process. Each entry is the flag associated with one block row. Then all the processes in the respective process row perform *only one* reduction operation on this flag array. This has greatly reduced the memory or communication latency cost. On the eight-core Clovertown, the improvement is significant, ranging from 6- to 9-fold. Even on the conventional SMP node, such as the eight-CPU Power5, we also obtained restpectable improvement, from 63% to 84%.

Note that the Clovertown time still does not scale as well as the Power5 time. Further investigation is needed in the future.

5 Final Remarks

We performed preliminary study of the SuperLU sparse direct solvers on representative multicore achitectures. Using the performance analysis tools such as PAPI and CrayPat, we gave quantitative measures of both static and temporal memory access bahavior. We found that the left-looking factorization incurs much less memory traffic than the right-looking one, therefore, it performs better on the CMP systems with limited memory bandwidth. We believe this performance characteristics is very likely associated with the other right-looking algorithm variants, such as a multifrontal algorithm. We also quantified that the arithmetic density of the triangular solution algorithm can be over an order

Table 8. `SuperLU_DIST` triangular solution time in seconds on Intel Clovertown and IBM Power5

matrix	tasks	Current				Improved			
		1	2	4	8	1	2	4	8
g7jac200	Clovertown	0.39	0.79	0.76	2.94	0.30	0.28	0.29	0.44
	Power5	0.61	0.68	0.46	0.39	0.43	0.39	0.28	0.22
stomach	Clovertown	0.93	1.21	3.79	6.74	0.77	0.74	0.53	0.90
	Power5	1.24	1.29	0.86	0.75	0.92	0.77	0.59	0.46
torso1	Clovertown	0.28	0.52	1.98	3.22	0.21	0.29	0.32	0.45
	Power5	0.31	0.41	0.27	0.24	0.22	0.24	0.18	0.13
twotone	Clovertown	0.46	1.51	4.42	7.52	0.32	0.44	0.47	0.80
	Power5	0.71	0.97	0.69	0.58	0.44	0.52	0.44	0.34

of magnitude lower than the preprocessing and factorization algorithms. The Pthreads code is usually more robust and delivers consistently better performance than the MPI code, particularly on a multicore+multithreading architecture, such as Sun VictoriaFalls. These suggest that it will be beneficial to use hybrid programming model, to design hybrid algorithms, and to provide hybrid device mode for MPICH.

In the future, we plan to continue using the performance tools to refine our understanding of multicore scaling, and find ways to enhance performance.

Acknowledgments

We used the multicore clusters with the PSI project and the RADlab at UC Berkeley, and the resources at the National Energy Research Scientific Computing Center. We are grateful to John Shalf, Rich Vuduc and Sam Williams for their help in using these machines and understanding the architectural features.

References

1. CrayPatCray Performance Analysis Tools, `http://docs.cray.com/books/S-2376-41/S-2376-41.pdf`
2. Davis, T.A.: University of Florida Sparse Matrix Collection, `http://www.cise.ufl.edu/research/sparse/matrices`
3. Demmel, J.W., Gilbert, J.R., Li, X.S.: An asynchronous parallel supernodal algorithm for sparse gaussian elimination. SIAM J. Matrix Analysis and Applications 20(4), 915–952 (1999)
4. Demmel, J.W., Gilbert, J.R., Li, X.S.: SuperLU Users Guide. Technical Report LBNL-44289, Lawrence Berkeley National Laboratory (September 1999)(Last update: September 2007), `http://crd.lbl.gov/~xiaoye/SuperLU/`
5. Duff, I.S., Koster, J.: 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. Li, X.S.: Sparse Gaussian elimination on high performance computers. Technical Report UCB//CSD-96-919, Computer Science Division, U.C. Berkeley, Ph.D dissertation (September 1996)

7. Li, X.S.: An overview of SuperLU: Algorithms, implementation, and user interface. ACM Trans. Mathematical Software 31(3), 302–325 (2005)
8. Li, X.S., Demmel, J.W.: SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. ACM Trans. Mathematical Software 29(2), 110–140 (2003)
9. MPICH - A Portable Implementation of MPI, `http://www-unix.mcs.anl.gov/mpi/mpich1/`
10. PAPI - Performance Application Programming Interface, `http://icl.cs.utk.edu/papi/`
11. Phillips, S.: Victoriafalls: Scaling highly-threaded processor cores. In: HOT CHIPS 19: A Symposium on High Performance Chips, Stanford, California, August 19-21 (2007)
12. Shalf, J.: Private communications
13. Williams, S.: Private communications
14. Williams, S., Oliker, L., Vuduc, R., Shalf, J., Yelick, K., Demmel, J.: Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In: Supercomputing (SC), Reno, California, November 10-16 (2007)

A Parallel Matrix Scaling Algorithm*

Patrick R. Amestoy[1], Iain S. Duff[2,3], Daniel Ruiz[1], and Bora Uçar[3]

[1] ENSEEIHT-IRIT, 2 rue Camichel, 31071, Toulouse, France
amestoy@enseeiht.fr, Daniel.Ruiz@enseeiht.fr
[2] Atlas Centre, RAL, Oxon, OX11 0QX, England
i.s.duff@rl.ac.uk
[3] CERFACS, 42 Av. G. Coriolis, 31057, Toulouse, France
duff@cerfacs.fr, ubora@cerfacs.fr

Abstract. We recently proposed an iterative procedure which asymptotically scales the rows and columns of a given matrix to one in a given norm. In this work, we briefly mention some of the properties of that algorithm and discuss its efficient parallelization. We report on a parallel performance study of our implementation on a few computing environments.

Keywords: Sparse matrices; matrix scaling; equilibration; parallel computing.

1 Introduction

Scaling a matrix consists of pre- and post-multiplying the original matrix by two diagonal matrices. We consider the following scaling problem: given a large, sparse matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, find two positive diagonal matrices $\mathbf{D}_1$ and $\mathbf{D}_2$ so that all rows and columns of the scaled matrix $\widehat{\mathbf{A}} = \mathbf{D}_1\mathbf{A}\mathbf{D}_2$ have the same magnitude in some norm. Two common choices for the norm are the ∞- and the 1-norm. Recently, we proposed an iterative algorithm for this purpose [16]. In this paper, we present the algorithm briefly and discuss how we parallelize it. We report experimental results with the parallel code on three parallel systems that have different processors and interconnection networks.

Scaling or equilibration of data for linear systems of equations is a topic of great importance that has already been the subject of several scientific publications, with many different developments depending on the properties required from the scaling. It has given rise to several well known algorithms; see, for example, [10,17]. If we denote by $\widehat{\mathbf{A}}$ the scaled matrix $\widehat{\mathbf{A}} = \mathbf{D}_1\mathbf{A}\mathbf{D}_2$, we then solve the equation $\widehat{\mathbf{A}}\hat{\mathbf{x}} = \hat{\mathbf{b}}$, where $\hat{\mathbf{x}} = \mathbf{D}_2^{-1}\mathbf{x}$ and $\hat{\mathbf{b}} = \mathbf{D}_1\mathbf{b}$.

A standard and well known approach to scaling is to do a row or column scaling. For row scaling, each row in the original matrix is divided by the norm of the row (using different norms, such as the ∞-norm or the 1-norm, depending on the

* This work was supported by "Agence Nationale de la Recherche" through Solstice project ANR-06-CIS6-010.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 301–313, 2008.

application). Column scaling is identical to row scaling, except that it considers the columns of the original matrix. A different approach that considers the matrix entries more globally is the one used in the HSL [13] routine MC29, which aims to make the nonzeros of the scaled matrix close to one by minimizing the sum of the squares of the logarithms of the moduli of the nonzeros in the scaled matrix; see [7]. MC29 reduces this sum in a global sense and therefore should be useful on a wide range of sparse matrices. There is also the routine MC30 in HSL that is a variant of the MC29 routine for symmetric matrices. Scaling can also be combined with permutations; see [11] and the HSL routine MC64. In this approach, the matrix is first permuted so that the product of absolute values of entries on the diagonal of the permuted matrix is maximized (other measures such as maximizing the minimum element are also options). Then the matrix is scaled so that the diagonal entries are one and the off-diagonals are less than or equal to one. This provides a useful preprocessing tool for pivoting for sparse direct solvers, as well as for building good preconditioners for an iterative method.

A good scaling will normally improve (i.e., reduce) the condition number of the matrix. Although this is not the whole story, for example in determining the efficacy of scaling for direct methods, this is a metric that we will later use to compare scaling algorithms.

The scaling algorithm and some of its properties are introduced in Section 2. We discuss our parallelization approach in Section 3. Section 4 contains the experimental results.

2 The Algorithm

Consider a general $m \times n$ real matrix $\mathbf{A}$, and denote by $\mathbf{r}_i = \mathbf{a}_{i\cdot}^T \in \mathbb{R}^{n\times 1}$, $i = 1, \ldots, m$, the row-vectors from $\mathbf{A}$ and by $\mathbf{c}_j = \mathbf{a}_{\cdot j} \in \mathbb{R}^{n\times 1}$, $j = 1, \ldots, n$, the column-vectors from $\mathbf{A}$. Denote by $\mathbf{D}_R$ and $\mathbf{D}_C$ the $m \times m$ and $n \times n$ diagonal matrices given by:

$$\mathbf{D}_R = \operatorname{diag}\left(\sqrt{\|\mathbf{r}_i\|_\infty}\right)_{i=1,\ldots,m} \text{ and } \mathbf{D}_C = \operatorname{diag}\left(\sqrt{\|\mathbf{c}_j\|_\infty}\right)_{j=1,\ldots,n} \tag{1}$$

where $\|\cdot\|_\infty$ stands for the ∞-norm of a real vector (that is the maximum entry in absolute value; sometimes called the max-norm). If a row (or a column) in $\mathbf{A}$ has all entries equal to zero, we replace the diagonal entry in $\mathbf{D}_R$ (or $\mathbf{D}_C$ respectively) by 1. In the following, we will assume that this does not happen, considering that such cases are fictitious in the sense that zero rows or columns should be taken away and the system reduced.

We then scale matrix $\mathbf{A}$ on both sides, forming the scaled matrix $\widehat{\mathbf{A}}$ in the following way

$$\widehat{\mathbf{A}} = \mathbf{D}_R^{-1}\mathbf{A}\mathbf{D}_C^{-1} \ . \tag{2}$$

The idea of the proposed algorithm is to iterate that process, resulting in Algorithm 1. Convergence is obtained when

$$\max_{1\le i\le m}\left\{|(1 - \|\mathbf{r}_i^{(k)}\|_\infty)|\right\} \le \varepsilon \text{ and } \max_{1\le j\le n}\left\{|(1 - \|\mathbf{c}_j^{(k)}\|_\infty)|\right\} \le \varepsilon \tag{3}$$

for a given value of $\varepsilon > 0$. We have shown [16] that the algorithm has fast linear convergence with an asymptotic rate of 1/2.

Algorithm 1. Simultaneous row and column iterative scaling in ∞-norm

1: $\mathbf{D}_1^{(0)} = \mathbf{I}_m$
2: $\mathbf{D}_2^{(0)} = \mathbf{I}_n$
3: **for** $k = 0, 1, 2, \ldots$ **until** convergence **do**
4: $\quad \mathbf{D}_R = \text{diag}\left(\sqrt{\|\mathbf{r}_i^{(k)}\|_\infty}\right)_{i=1,\ldots,m}$
5: $\quad \mathbf{D}_C = \text{diag}\left(\sqrt{\|\mathbf{c}_j^{(k)}\|_\infty}\right)_{j=1,\ldots,n}$
6: $\quad \mathbf{D}_1^{(k+1)} = \mathbf{D}_1^{(k)}\,\mathbf{D}_R^{-1}$
7: $\quad \mathbf{D}_2^{(k+1)} = \mathbf{D}_2^{(k)}\,\mathbf{D}_C^{-1}$
8: $\quad \widehat{\mathbf{A}}^{(k+1)} = \mathbf{D}_1^{(k+1)}\mathbf{A}\mathbf{D}_2^{(k+1)}$

For nonnegative square matrices, using the 1-norm, instead of the ∞-norm in lines 4 and 5 results in a scaling algorithm for the 1-norm, i.e., in the scaled matrix, the 1-norm of each row and column is asymptotically equal to 1. Convergence in the 1-norm case of both $\mathbf{A}^{(k)}$ and $\mathbf{D}_1^{(k)}$ and $\mathbf{D}_2^{(k)}$ is guaranteed for nonnegative matrices with total support—a square matrix is said to have total support if all entries can appear in some zero-free diagonal after row or column permutations. If a matrix does not have total support but just support (i.e., there exists a zero-free diagonal after row or column permutations), then the algorithm converges only for the $\mathbf{A}^{(k)}$ iterates; see [16] for details. We have observed in practical experiments that convergence for the 1-norm is fast for matrices with total support; for matrices with support but without total support, some entries should asymptotically go to zero, and a painfully slow convergence can be observed. Rothblum et al. have shown [15, page 13] that the problem of scaling a matrix $\mathbf{A}$ in the l_p-norm, $1 < p < \infty$ can be reduced to the problem of scaling in the 1-norm the pth Hadamard power of $\mathbf{A}$, i.e.,the matrix $\mathbf{A}^{[p]} = [a_{ij}^p]$. We applied that discussion to Algorithm 1 by replacing the matrix $\mathbf{A}$ with $\mathbf{A}^{[p]}$ and by taking the Hadamard pth root, e.g., $\mathbf{D}_1^{[1/p]} = [d_{ii}^{1/p}]$, of the resulting iterates. Hence, we argue that all of the convergence results that hold for the 1-norm hold for any of the l_p norms for $1 < p < \infty$.

We emphasize that the proposed iterative scaling procedure preserves the symmetry of the original matrix. If the given matrix $\mathbf{A}$ is symmetric, then the diagonal matrices $\mathbf{D}_R$ and $\mathbf{D}_C$ in Eq. (1) are equal and, consequently, matrix $\widehat{\mathbf{A}}$ in Eq. (2) is symmetric, as is the case for the matrices $\widehat{\mathbf{A}}^{(k)}$ at any iteration in Algorithm 1. This is not the case for most scaling algorithms which alternately scale rows followed by columns or vice-versa.

In the case of unsymmetric matrices, one may consider the use of the Sinkhorn-Knopp iterations [18] with the ∞-norm in place of the 1-norm. This method simply normalizes all rows and then columns in $\mathbf{A}$, and iterates on this process until

convergence. In the ∞-norm, this is obtained after a single step. Because of its simplicity, this method is very appealing. Notice, however, that the Sinkhorn-Knopp iteration may provide very different results when applied to $\mathbf{A}$ or $\mathbf{A}^T$. On the contrary, Algorithm 1 provides exactly the same results when applied to $\mathbf{A}$ or $\mathbf{A}^T$ in the sense that the scaled matrix obtained from $\mathbf{A}^T$ is the transpose of that obtained from $\mathbf{A}$. Another related property of Algorithm 1 is that it is independent of matrix permutations. In other words, the scaling factors of the permuted matrix are equivalent to the permuted scaling factors of the original matrix.

3 Parallelization

Algorithm 1 involves the scaled matrix $\widehat{\mathbf{A}}^{(k)}$, the original matrix $\mathbf{A}$, the two scaling (diagonal) matrices $\mathbf{D}_1^{(k)}$ and $\mathbf{D}_2^{(k)}$, and two temporary (diagonal) matrices $\mathbf{D}_R$ and $\mathbf{D}_C$ to compute the next iterates. To reduce the memory requirements, it is advisable not to store the scaled matrix $\widehat{\mathbf{A}}^{(k)} = \mathbf{D}_1^{(k)}\mathbf{A}\mathbf{D}_2^{(k)}$ explicitly; an individual matrix entry $a_{ij}^{(k)}$ at iteration k can be computed using $d_1^{(k)}(i) \times |a_{ij}| \times d_2^{(k)}(j)$, where $d_1^{(k)}(i)$ and $d_2^{(k)}(j)$ correspond to the ith and jth diagonal entries of the respective scaling matrices. Therefore, a parallelization of the algorithm on distributed memory processors necessitates the distribution of the matrices $\mathbf{A}$, $\mathbf{D}_1$, $\mathbf{D}_2$, $\mathbf{D}_R$ and $\mathbf{D}_C$. Observe that $\mathbf{D}_1$ and $\mathbf{D}_2$ are kept and updated at each iteration, whereas $\mathbf{D}_R$ and $\mathbf{D}_C$ are computed afresh at every iteration.

Assume that the matrix $\mathbf{A}$ is distributed among P processors. At this point we do not assume a particular distribution. Rather, we deal with the most general case in which each processor holds a set of nonzeros a_{ij} along with the corresponding row and column indices, i.e., each processor holds a set of triplets of the form $\langle i, j, a_{ij}\rangle$. We use $a_{ij} \in p$ to denote that the processor p has the nonzero a_{ij}. At each iteration, we first compute the contribution to the matrices $\mathbf{D}_R$ and $\mathbf{D}_C$ on each processor, using $\mathbf{D}_R^p$ and $\mathbf{D}_C^p$—the latter two matrices denote the matrices belonging to the processor p such that $d_R^p(i)$ and $d_C^p(j)$ denote the contributions of processor p to $d_R(i)$ and $d_C(j)$, respectively. These two matrices are then reduced to update the diagonal matrices $\mathbf{D}_1$ and $\mathbf{D}_2$ that are distributed among the processors; i.e., the partial results $d_R^p(i)$ and $d_C^p(j)$ should be combined at certain processors according to the partitions on $\mathbf{D}_1$ and $\mathbf{D}_2$. Hence, our problem reduces to partitioning the diagonal matrices $\mathbf{D}_1$ and $\mathbf{D}_2$ for a given partition on $\mathbf{A}$ in order to have an efficient parallelization of Algorithm 1. The most common communication cost metric addressed in similar parallelization problems is the total communication volume. Therefore our aim is to find partitions on $\mathbf{D}_1$ and $\mathbf{D}_2$ for a given partition on $\mathbf{A}$ to minimize the total communication volume.

In order to solve the partitioning problem, let us examine the computational dependencies. Each processor p should use its triplets $\langle i, j, a_{ij}\rangle$ to compute partial results for $d_R(i)$ and $d_C(j)$, e.g., for the ∞-norm processor p should compute

$$d_R^p(i) = \max_j \left\{ d_1^{(k)}(i) \times |a_{ij}| \times d_2^{(k)}(j) : a_{ij} \in p \right\} .$$

The partial results should be reduced for each $d_1^{(k+1)}(i)$ and $d_2^{(k+1)}(j)$, e.g., in the ∞-norm the owner of $d_1(i)$ should compute

$$d_1^{(k+1)}(i) = d_1^{(k)}(i) \times \frac{1}{\sqrt{\max\{d_R^p(i) : 1 \leq p \leq P\}}} .$$

Note that the communication operations take place during these reduction operations. That is, the partial results $d_R^p(i)$ from each processor p, where $1 \leq p \leq P$ and there exist a $a_{ik} \in p$ for some k, should be sent to the processor which is responsible for computing $d_1^{(k+1)}(i)$. After computing $d_1^{(k+1)}(i)$, the owner should send the new values back to the contributing processors to enable the computation of $\widehat{\mathbf{A}}^{(k+1)}$. That is, the owner sends the updated $d_1^{(k+1)}(i)$ to each processor p having a nonzero in row i, e.g., to a processor p where $a_{ik} \in p$ for some k. Therefore, the volume of data a processor receives to compute $d_1^{(k+1)}(i)$ is equal to the volume of data it sends after computing the final value.

If the nonzeros in row $\mathbf{r}_i$ are split among s processors, then a reduction on s partial results will be necessary. If one of those processors owns $d_1(i)$, then $s-1$ partial results will be sent to the owner; if not, then s partial results will be sent to the owner. Hence, for a given partition on $\mathbf{A}$, the minimum volume of communication regarding $\mathbf{r}_i$ is $s-1$. The same assertions hold for $d_2(j)$ with respect to the nonzeros in column $\mathbf{c}_j$. Therefore, if the nonzeros in row $\mathbf{r}_i$ and column $\mathbf{c}_j$ are split among $s_r(i)$ and $s_c(j)$ processors, respectively, then the minimum total communication volume is

$$2 \times \sum (s_r(i) - 1) + 2 \times \sum (s_c(j) - 1) , \tag{4}$$

where half of the communication volume is incurred while reducing the new values and the other half is incurred while sending back the updated values. The minimum total communication volume can be achieved for any partition on $\mathbf{A}$ as long as each $d_1(i)$ and $d_2(j)$ are assigned to a processor which has nonzeros in row $\mathbf{r}_i$ and column $\mathbf{c}_j$, respectively. Furthermore, any $d_1(i)$ to processor p (or $d_2(j)$ to processor p) assignment will attain the same minimum as long as the processor p has at least one nonzero in row $\mathbf{r}_i$ (or column $\mathbf{c}_j$).

It can be seen from Eq. (4) that the communication volume requirements of the proposed algorithm are closely related to those of repeated sparse matrix-vector multiply operations; see for example [4,12]. In fact, the communication operations in an iteration of Algorithm 1 are the same as those in the computations $\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}$ followed by $\mathbf{x} \leftarrow \mathbf{A}^T\mathbf{y}$, when the partitions on $\mathbf{x}$ and $\mathbf{y}$ are equal to the partitions on $\mathbf{D}_2$ and $\mathbf{D}_1$, respectively. Having observed that, we can use hypergraph models, see for example [4,20,22], to partition the matrix $\mathbf{A}$, and then follow the above development to partition $\mathbf{D}_1$ and $\mathbf{D}_2$ to achieve efficient parallelization. Moreover, due to the equivalence between the communication operations of the proposed algorithm and those of sparse matrix-vector multiply operations, we can adopt the vector partitioning techniques discussed in [1,21] to partition $\mathbf{D}_1$ and $\mathbf{D}_2$.

We wanted to have a parallelization of the scaling algorithm independent of the matrix partitioning. This is because we imagine the use of the algorithm in

a parallel linear system solver context where the matrix is already distributed. Therefore, as an alternative to the existing partitioning methods [1,21], we developed the following parallel algorithm to partition $\mathbf{D}_1$ and $\mathbf{D}_2$ among P processors. In the conceived partitioning although, each $d_1(i)$ will be assigned to the processor which has the closest entry to a fictitious diagonal (on a square matrix of order $\max\{m, n\}$). The same strategy is used for each $d_2(j)$ with respect to the columns. If, for example, the matrix $\mathbf{A}$ is square and has a zero-free diagonal, then the proposed approach will partition $\mathbf{D}_1$ and $\mathbf{D}_2$ in such a way that the processor which holds a_{ii} will own $d_1(i)$ and $d_2(i)$. This is the common approach taken in standard matrix partitioning approaches, see for example [4,6]. The MPI standard [14] defines an operation, `minloc`, which can be used to accomplish this task. In general, `minloc` can be used to compute a global minimum and the rank of the process whose data contain that minimum value. Below, we discuss how we went about implementing this operation for our partitioning method (essentially the same as `minloc` operation).

In our implementation, we perform a reduction operation on two arrays of sizes $2 \times m$ and $2 \times n$. Each processor p initializes a local copy of both arrays by setting $g_r^p(i) = m + n$ for $1 \leq i \leq m$ and $g_r^p(i) = p$ for $i > m$, and similarly $g_c^p(j) = m + n$ for $1 \leq j \leq n$ and $g_c^p(j) = p$. Then, each processor sweeps over its triplets $\langle i, j, a_{ij} \rangle$ and computes its shortest distance to the diagonal entry in row i and its shortest distance to the diagonal entry in column j. That is, processor p computes

$$g_r^p(i) = \min\{|i - j| : a_{ij} \in p\}$$

and

$$g_c^p(j) = \min\{|j - i| : a_{ij} \in p\} \; .$$

As we see, at the end of these operations, the shortest distances are stored in the first halves of the arrays whereas the second halves store the rank of the associated processor. A global all-reduce operation is performed on these two arrays to yield an array g_r of size $2 \times m$ and another one g_c of size $2 \times n$ on all processors. The reduction operation is performed with the minimum operation to set

$$g_r(i) = \min_p \{g_r^p(i) : 1 \leq p \leq P\} \;\text{ for }\; 1 \leq i \leq m$$

and

$$g_c(j) = \min_p \{g_c^p(j) : 1 \leq p \leq P\} \;\text{ for }\; 1 \leq j \leq n \; .$$

The second halves of the arrays are used to store the ranks of the processor that give the minimum value stored in the corresponding entries in the first halves of the arrays. The second halves are necessary to guarantee a unique partitioning vector on all processors. If there is a tie for an entry in the first half of the arrays, the processor with the smaller rank is declared as the one giving the minimum—a unique partitioning vector is not possible without further communication if, for example, the second halves were not used and the ties were broken at random. Note that, although different implementations are possible,

MPI's `minloc` operation also uses an array of size twice the number of data items to store the rank of the processor that owns a copy of the minimum value.

We make a few observations about the proposed partitioning algorithm. Firstly, the proposed diagonal matrix partitioning approach tries to exploit the given partition of the matrix $\mathbf{A}$ and obtains the minimum total volume of communication possible (with respect to the given partition on $\mathbf{A}$). This is achieved by assigning each $d_1(i)$ or $d_2(j)$ to a processor having nonzeros entries in the respective row or column of the matrix. In other words, the proposed algorithm achieves the minimum given in Eq. (4); if the matrix is partitioned with the objective of minimizing the total communication volume, then the total communication volume found there is retained intact by the algorithm. Secondly, we believe that the algorithm is likely to achieve a balance on the number of $\mathbf{D}_1$ and $\mathbf{D}_2$ matrix entries assigned to the processors, hence in a way it will achieve a balance on communication loads of the processors. We investigate the issue of the balance achieved in the communication loads of the processors in the next section. We note that the problem of optimizing the partitioning of $\mathbf{D}_1$ and $\mathbf{D}_2$ for some other communication cost metrics such as the total number of messages with a balancing constraint on the communication volume loads of processors, or the maximum volume of messages sent and received by a single processor is NP-complete; see [20] and [1], respectively. Rather than addressing such communication cost metrics explicitly, we prefer the proposed partitioning algorithm, as it is easy to implement and is fast to run in parallel.

4 Experiments

We have implemented a parallel program for the proposed matrix scaling algorithm in C using LAM/MPI [3]. The experiments were carried out on up to 16 nodes of two PC clusters of Beowulf class [19]. In the first cluster, the nodes are Intel Pentium IV 2.6 GHz processors with 1GB of RAM, and they run Debian/GNU Linux. This cluster has a Gigabit Ethernet switch. The cluster has a measured latency of 37 microseconds and a measured bandwidth of 75MB/s. The second cluster has an Infiniband interconnection network and is based on Dual 250 Opteron AMD processors each having 4GB of RAM. In this cluster, latency and bandwidth are measured as 3.3 microseconds and 772MB/s, respectively. In both of the systems, the program is compiled with `gcc` using the optimization option `-O3`.

We ran the program on a set of matrices from the University of Florida sparse matrix collection [8]. The characteristics of the matrices are shown in Table 1.

We have observed that usually 25–30 iterations of the discussed scaling algorithm is sufficient to improve the condition number of the matrices. We used the performance profiles discussed in [9] to generate the plot shown in Fig. 1. The plot compares estimates of the condition numbers for the scaled matrices resulting from four different scaling algorithms and those of the original matrices. For a given τ, the plot shows the probability for a scaling algorithm that the condition estimate due to this algorithm is within τ times the best (among all 5 condition

Table 1. Matrices used in measuring the parallel performance, their size, number of nonzeros, and the number of iterations to converge in the ∞- and 1-norms with error tolerance of 1.0e-6. The number 1000 indicates cases where the method did not converge in 1000 iterations (those matrices, except `Hamrle3`, do not have total support). Matrices are listed in increasing order of the number of nonzeros.

			number of iterations	
matrix	n	nnz	∞-norm	1-norm
aug3dcqp	35543	128115	26	50
a5esindl	60008	255004	2	107
a2nnsnsl	80016	355034	22	115
a0nsdsil	80016	355034	22	106
blockqp1	60012	640033	2	48
olesnik0	88263	744216	23	1000
c-71	76638	859554	24	1000
boyd1	93279	1211231	25	28
twotone	120750	1224224	24	1000
lhr71c	70304	1528092	27	1000
H2O	67024	2283760	2	16
filter3D	106437	2813616	3	20
Hamrle3	1447360	5514242	23	1000
G3_circuit	1585478	9246304	2	19
thermal2	1228045	9808358	2	18
SiO2	155331	11438834	2	16

estimates). Therefore, the higher the probability the more preferable the scaling method. We have plotted the performance profiles up to $\tau = 5$. As seen in the plot, the condition estimate of the original matrix has the worst profile; at any τ, the condition estimate of the original matrix has the least probability to be the best. As also seen from the plot, the discussed scaling algorithm with any of the norms (1-, 2-, or ∞) has higher probability to be better than that of Bunch's for τ a little larger than 1.5. We note that Bunch's algorithm is a direct approach; it computes the scaling factors without any iterations. Note that for these results we run the parallel scaling algorithms for at most 25 iterations. Although the 1- and 2-norm scaling algorithms did not fully converge in 136 of the 245 cases, the resulting condition estimates after 25 iterations were close to the best value, e.g., with a high probability, the results are within a small τ of the best. The Sinkhorn-Knopp algorithm gave almost the same condition estimates as the parallel scaling algorithm with the 1- and 2-norms.

To measure the average running time of an iteration, we ran the program for 1000 iterations, without testing convergence. We used the fine-grain hypergraph model [6] and the hypergraph partitioning tool PaToH [5] with default options to partition the matrices. In the fine-grain model, the nonzeros of a matrix are partitioned independently, i.e., nonzeros in a row or a column are not necessarily assigned to a common processor. We compute the partitions on $\mathbf{D}_1$ and $\mathbf{D}_2$ with the parallel algorithm proposed towards the end of Section 3.

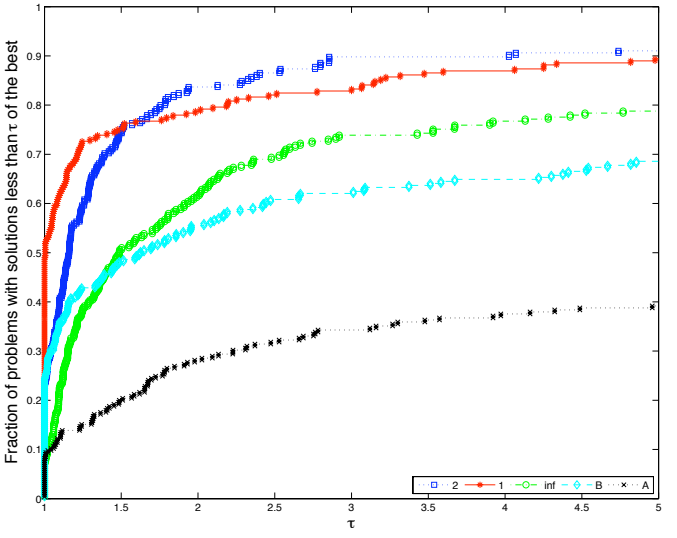


Fig. 1. Performance profiles for the condition number estimates for 245 matrices. `A` marks the condition number estimate of the original matrix; `B` marks that of Bunch's algorithm [2]; inf, 1, and 2 mark that of the parallel scaling algorithm with ∞-, 1-, and 2-norms (with at most 25 iterations). At, for example $\tau = 3$, the curves from top to bottom correspond to the labels given in the legend from left to right.

Table 2 shows the speedups we have obtained for the matrices in our data set. Note that we measure the time spent in the iterations, and hence assume that the matrix is already distributed among the processors. During these experiments, the convergence tests are not performed, and hence the reported time of the iterations does not include the time spent doing the convergence checks. The speedups are the averages of 10 different matrix partitions obtained with the fine-grain model. As is seen in the table, good results are obtained for the bigger (in terms of number of nonzeros) matrices (except for `c-71` and `H2O`). That is, most of the time, we obtain better speedups for matrices with a larger number of nonzeros. This is expected as the computation to communication ratio is small for sparse matrix-vector multiply type operations. Therefore, if the matrix has a small number of nonzeros, the communication overhead becomes significant and degrades the performance. We investigated the communication patterns in an attempt to understand the performance of the proposed parallelization approach. Notice that the load balance and the total communication volume are determined according to the given matrix distribution. In all cases, the load imbalance was less than 0.03; we measure the imbalance as $(w_{max} - w_{avg})/w_{avg}$, where w_{max} is the maximum load and w_{avg} is total load divided by the number processors, so the value zero would indicate perfect balance, a value of 1 that the maximum load was twice the average and a value greater than 1 would indicate severe imbalance. The algorithm proposed for partitioning $\mathbf{D}_1$ and $\mathbf{D}_2$ resulted in acceptable imbalances among the communication loads of the processors. In

Table 2. Speedup values of the parallel scaling algorithm with ∞-norm, on $P = 2, 4, 8$, and 16 processors for two different parallel systems. For each matrix, the first and second lines correspond to the experiments run on, respectively, PC cluster with Intel processors and PC cluster with AMD processors. For each matrix, the sequential running time of the scaling algorithm for 1000 iterations is listed in units of seconds under the column Seq.Time.

		P			
matrix	Seq.Time	2	4	8	16
aug3dcqp	8.30	1.7	2.9	4.1	4.5
	3.06	1.9	3.8	4.3	3.6
a5esindl	15.09	1.8	3.0	4.1	4.8
	5.12	1.5	1.9	2.3	3.8
a2nnsnsl	20.71	1.8	3.1	4.0	4.8
	7.24	1.5	1.8	2.1	3.3
a0nsdsil	20.92	1.8	3.1	4.0	4.6
	7.22	1.5	1.8	2.1	3.2
blockqp1	32.55	1.9	3.4	5.5	7.4
	8.97	1.6	2.4	3.3	4.9
olesnik0	46.08	1.9	3.7	6.9	12.3
	14.91	1.9	3.9	7.5	13.6
c-71	51.60	1.8	3.3	5.4	7.6
	17.54	1.6	3.3	5.3	6.7
boyd1	70.34	1.9	3.6	6.3	10.2
	24.57	1.8	3.1	4.9	7.6
twotone	74.76	1.9	3.7	7.0	11.8
	25.40	1.9	3.7	6.9	11.3
lhr71	78.25	2.0	3.8	7.3	13.5
	18.10	2.0	3.4	6.8	14.0
H2O	111.33	1.9	2.8	2.4	6.7
	29.33	1.6	2.5	4.2	7.7
filter3D	146.83	1.9	3.7	7.1	13.3
	52.66	2.1	3.5	6.7	12.7
Hamrle3	337.99	1.9	3.8	7.3	13.9
	146.15	1.9	3.8	7.0	12.6
G3_circuit	455.25	1.8	3.8	7.4	14.0
	173.11	1.9	3.3	6.9	14.5
thermal2	573.24	2.0	3.9	7.6	14.4
	208.20	1.6	3.4	6.5	13.1
SiO2	545.90	1.9	3.7	6.9	11.3
	180.09	1.9	3.6	5.9	9.5

terms of number messages sent by a single processor, the imbalance among loads of the processors, is on the average, 0.25 with a maximum of 1.45. In terms of volume of messages sent by a single processor, the average imbalance is 0.4, with a maximum of 3.27. We further investigated the communication patterns for 64- and 128-way partitions of the matrices in our data set. Although we have seen some large numbers, the average imbalance is around 3.2 using a metric of the

maximum number of messages per processor, and 4.3 with the metric being the maximum volume of messages per processor.

In an attempt to verify empirically that the proposed algorithm for partitioning $\mathbf{D}_1$ and $\mathbf{D}_2$ works well for a number of systems, we performed experiments on the nodes of a CRAY XD1 system at CERFACS. This system has two AMD Opteron 2.4 GHz processors per node, each having 2GBytes of memory. The nodes are connected with a RapidArray interconnect with an MPI latency of 1.7 microseconds and a bandwidth of 4GB/s between nodes. The speedups obtained in this system are similar to the reported results.

We have also investigated a sensible alternative partitioning approach on the three parallel systems mentioned so far. The alternative approach is to assign $d_1(i)$ to the processor with the smallest rank among those having nonzeros in row $\mathbf{r}_i$; assign $d_2(j)$ to the processor with the largest rank among those having nonzeros in column $\mathbf{c}_j$. Note that this alternative will also achieve the same minimum in the total volume of communication metric, see Eq. (4). On the PC cluster with AMD processors and Infiniband interconnect and also on the CRAY XD1, the use of this alternative resulted in speedups similar to those resulting from the proposed partitioning approach. However, the alternative did not perform as well on the PC cluster mentioned before. Note that the alternative can produce high imbalance among the number of messages sent by a single processor. Furthermore, the messages are usually short. Combined with the relatively high message latency overhead, this is the most probable reason behind the PC cluster being intolerant to simple partitioning algorithms. In fact, we have observed that the alternative resulted in imbalances, on the average, of around 1.0 for the communication cost metrics of number of messages and communication volume per processor, both in terms of sends and receives, with the maximum being 7.0 for all of the metrics, which is really a very bad imbalance.

5 Conclusion

In this work, we reviewed an iterative algorithm which scales the l-norm, for $l = 1, 2, \ldots, \infty$, of the rows and columns of a matrix to 1 and briefly mentioned some of its properties. We discussed the parallelization of the algorithm. We argued that the parallelization requires a careful partitioning of two diagonal matrices in addition to a standard sparse matrix partitioning for parallel matrix-vector multiply operations. We proposed a method based on an all-reduce operation to partition the diagonal matrices. We discussed performance results on different parallel systems where good speedups are obtained for matrices having a reasonably large number of nonzeros.

Acknowledgements

We thank Prof. C. Aykanat and members of the parallel and distributed computing research group of Bilkent University, Ankara, Turkey; and Prof. Ü. V. Çatalyürek of the Department of Biomedical Informatics at the Ohio

State University, USA for granting us exclusive access to their parallel machines. We also thank Dr. Jean-Yves L'Excellent of Laboratoire de l'Informatique du Parallélisme, ENS Lyon, France for pointing out the MPI's `minloc` operation.

References

1. Bisseling, R.H., Meesen, W.: Communication balancing in parallel sparse matrix-vector multiplication. Electronic Transactions on Numerical Analysis 21, 47–65 (2005)
2. Bunch, J.R.: Equilibration of symmetric matrices in the max-norm. Journal of the ACM 18(4), 566–572 (1971)
3. Burns, G., Daoud, R., Vaigl, J.: LAM: an open cluster environment for MPI. In: Ross, J.W. (ed.) Proceedings of Supercomputing Symposium 1994, pp. 379–386. University of Toronto (1994)
4. Çatalyürek, Ü.V., Aykanat, C.: Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. IEEE Transactions on Parallel and Distributed Systems 10(7), 673–693 (1999)
5. Çatalyürek, Ü.V., Aykanat, C.: PaToH: A multilevel hypergraph partitioning tool, version 3.0. Technical Report BU-CE-9915, Computer Engineering Department, Bilkent University (1999)
6. Çatalyürek, Ü.V., Aykanat, C.: A fine-grain hypergraph model for 2d decomposition of sparse matrices. In: Proceedings of 15th International Parallel and Distributed Processing Symposium (IPDPS), San Francisco, CA (April 2001)
7. Curtis, A.R., Reid, J.K.: On the automatic scaling of matrices for Gaussian elimination. IMA Journal of Applied Mathematics 10(1), 118–124 (1972)
8. Davis, T.A.: University of Florida sparse matrix collection. NA Digest, 92/96/97, 1994/1996/1997, `http://www.cise.ufl.edu/research/sparse/matrices`
9. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Mathematical Programming 91(2), 201–213 (2002)
10. Duff, I.S., Erisman, A.M., Reid, J.K.: Direct Methods for Sparse Matrices. Oxford University Press, London (1986)
11. Duff, I.S., Koster, J.: 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)
12. Hendrickson, B., Kolda, T.G.: Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing. SIAM Journal on Scientific Computing 21(6), 2048–2072 (2000)
13. HSL: A collection of Fortran codes for large-scale scientific computation (2004), `http://www.cse.scitech.ac.uk/nag/hsl`
14. MPI: A Message-Passing Interface Standard, Version 2.1 (2008), `http://www.mpi-forum.org/docs/`
15. Rothblum, U.G., Schneider, H., Schneider, M.H.: Scaling matrices to prescribed row and column maxima. SIAM Journal on Matrix Analysis and Applications 15(1), 1–14 (1994)
16. Ruiz, D.: A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical Report RAL-TR-2001-034 and RT/APO/01/4, Rutherford Appleton Laboratory, Oxon, UK and ENSEEIHT-IRIT, Toulouse, France (2001)
17. Schneider, M.H., Zenios, S.: A comparative study of algorithms for matrix balancing. Operations Research 38(3), 439–455 (1990)

18. Sinkhorn, R., Knopp, P.: Concerning nonnegative matrices and doubly stochastic matrices. Pacific Journal of Mathematics 21(2), 343–348 (1967)
19. Sterling, T., Savarese, D., Becker, D.J., Dorband, J.E., Ranaweke, U.A., Packer, C.V.: BEOWULF: A parallel workstation for scientific computation. In: Proceedings of the 24th International Conference on Parallel Processing (1995)
20. Uçar, B., Aykanat, C.: Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. SIAM Journal on Scientific Computing 25(6), 1827–1859 (2004)
21. Uçar, B., Aykanat, C.: Revisiting hypergraph models for sparse matrix partitioning. SIAM Review 49(4), 595–603 (2007)
22. Vastenhouw, B., Bisseling, R.H.: A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. SIAM Review 47(1), 67–95 (2005)

Design, Tuning and Evaluation of Parallel Multilevel ILU Preconditioners

José I. Aliaga[1], Matthias Bollhöfer[2], Alberto F. Martín[1], and Enrique S. Quintana-Ortí[1]

[1] Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain, Tel.: (+34) 964-728278, Fax: (+34) 964-728486 {aliaga,martina,quintana}@icc.uji.es.

[2] Institute of Computational Mathematics, TU-Braunschweig, D-38106 Braunschweig, Germany, Tel.: (+49) 531-391-7536, Fax: (+49) 531-391-8206 m.bollhoefer@tu-braunschweig.de.

Abstract. In this paper, we present a parallel multilevel ILU preconditioner implemented with OpenMP. We employ METIS partitioning algorithms to decompose the computation into concurrent tasks, which are then scheduled to threads. Concretely, we combine decompositions which obtain significantly more tasks than processors, and the use of dynamic scheduling strategies in order to reduce the thread's idle time, which it is shown to be the main source of overhead in our parallel algorithm. Experimental results on a shared-memory platform consisting of 16 processors report remarkable performance for our approach.

Keywords: Sparse linear system, incomplete LU factorization, parallel algorithm, OpenMP, shared-memory multiprocessor.

Related conference topics: Parallel and Distributing Computing, Numerical Algorithms for CS&E.

1 Introduction

The solution of large sparse linear systems is an ubiquitous problem in chemistry, physics, and engineering applications. Often, sparse direct solvers are used to deal with these problems, but the large amount of memory they sometimes require (due to, e.g., excessive fill-in in the factors), in practice limits the size of the problems these methods can solve. In recent years, iterative methods based on Krylov subspaces combined with preconditioners as, e.g., incomplete LU decompositions, have become popular and successful in many application problems. Among these methods, ILUPACK[1] (*Incomplete LU decomposition PACKage*) is a novel software package based on approximate factorizations which enhances the performance of the process in terms of a more accurate solution and a lower execution time.

[1] http://www.math.tu-berlin.de/ilupack

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 314–327, 2008.

Our mid-term goal is to develop a parallel package to solve large sparse linear systems on shared-memory multiprocessors (including novel multicore processors) using the same techniques employed in ILUPACK. In an earlier paper [1], we presented a parallel preconditioner for the solution of sparse linear systems with symmetric positive definite coefficient matrix, and an OpenMP-based implementation of this preconditioner. In this paper, we extend previous work with the following new contributions:

- We discuss in detail the foundations of the design of the parallel preconditioner.
- We evaluate the preconditioner using a benchmark collection with problems arising from different domains such as, e.g., computational fluid dynamics (CFD) or circuit simulation.
- We split the task tree deeply in order to decompose the computation into a large number of fine-grain tasks.
- We provide a detailed explanation of the experimental performance. In particular, we use the KOJAK[2] and VAMPIR[3] performance analysis tools to gain a complete understanding of the performance of the parallel algorithm.

The paper is structured as follows. In Section 2 we briefly review the basis of the preconditioning procedure underlying ILUPACK. Next, in Section 3, we offer further details on the design of our parallel preconditioner. Section 4 gathers data from numerical experiments with our parallel algorithm implementation. Concluding remarks and future research goals follow in Section 5.

2 An Overview of ILUPACK

ILUPACK is a preconditioning package to solve large sparse linear systems via preconditioned Krylov subspace methods, where the preconditioner is based on multilevel ILUs. We will focus on the computation of the preconditioner since this is the most challenging task from the parallelization viewpoint.

The rationale behind the computation of the preconditioner is to obtain an incomplete LU decomposition of the coefficient matrix A in such a way that element growth in the inverse triangular factors remains bounded, thereby improving the quality of the preconditioner [3,4,5,14]. ILUPACK adopts the following combination of steps in a multilevel manner in pursue of this goal:

1. A static pre-ordering and scaling of the system to transform A as
$$A \to P^T D_1 A D_2 Q = \hat{A}. \tag{1}$$
The reordering strategy is intended to reduce the fill-in, while scaling is applied in order to balance the size of the entries.
2. A partial incomplete LU factorization of $\hat{A}$ is applied, where rows/columns associated with "tiny" diagonal pivots are permuted to the bottom right corner of the matrix; see Figure 1. For the multilevel ILU, a bound κ for the

[2] `http://www.fz-juelich.de/jsc/kojak/`
[3] `http://www.vampir-ng.de`

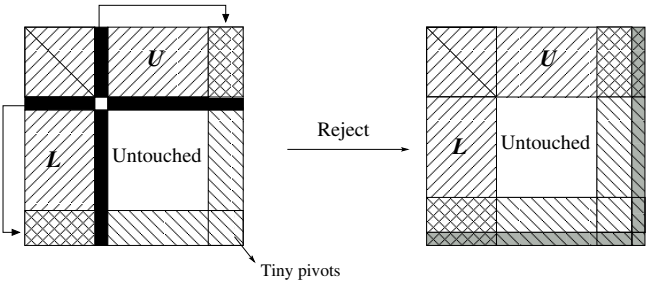


Fig. 1. ILUPACK pivoting strategy

(norm of the) inverse triangular factors is prescribed. Tiny pivots are thus defined as those which cause these inverses to exceed κ. These rows/columns are not considered further in this step: their decomposition is delayed until the next level. Thus, we obtain a partial approximation of the following reordered system

$$\hat{P}^T \hat{A} \hat{P} = \begin{pmatrix} B & F \\ E & C \end{pmatrix} \approx \begin{pmatrix} L_B & 0 \\ L_E & I \end{pmatrix} \begin{pmatrix} D_B & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} U_B & U_F \\ 0 & I \end{pmatrix}, \tag{2}$$

where L_B, U_B^T are unit lower triangular factors, D_B is a diagonal matrix, and S is the approximate Schur complement, where tiny pivots reside. The Schur complement corresponds to:

$$S \approx C - L_E D_B U_F. \tag{3}$$

Elements of magnitude ϵ/κ are dropped from the factors as well as from S during the computation, where ϵ is a user-defined tolerance. For details, see [5].

3. A multilevel configuration, which recursively applies the previous two steps to S, completes the partial decomposition of step 2.

3 Parallel Multilevel Preconditioners

3.1 Task Decomposition

The first step in the development of a parallel preconditioner consists of splitting the procedure that computes the preconditioner into tasks, and identifying the dependencies among these. In the context of the sparse direct solvers, elimination trees are widely used as a source of coarse-grain parallelism [8]. The starting point of our approach is the *elimination tree* [7] of a sparse symmetric matrix ordering. Thus, e.g., if T_i and T_j are independent subtrees of the elimination tree (i.e., neither root node of the subtrees is a descendant of the other), then the operations on the rows corresponding to the nodes in T_i can

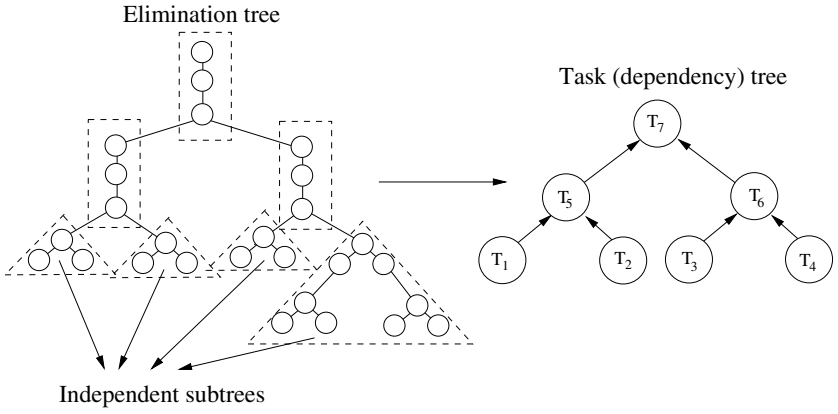


Fig. 2. Elimination tree and an associated task tree of height 2

proceed independently from those corresponding to the nodes in T_j. Hence these computations can be done simultaneously by separate processors with no communication between them. Therefore, we can define a new tree, which we call the *task (dependency) tree*, where the leaf tasks correspond to the independent subtrees, and the intermediate dependent tasks correspond to the ancestor nodes of these subtrees in the elimination tree; see Figure 2.

These abstractions reveal the parallelism in the sparse Cholesky factorization, and can also be used to obtain a multilevel ILU preconditioner in parallel. We now discuss how this preconditioner is computed using the task tree in Figure 2 (right). We first note that this task tree yields a partition of the coefficient matrix A. For the discussion, we consider the matrix reordering which corresponds to a breadth-first traversal of the task tree from bottom to top:

$$A \rightarrow \left(\begin{array}{cccc|cc|c} A_{11} & 0 & 0 & 0 & A_{15} & 0 & A_{17} \\ 0 & A_{22} & 0 & 0 & A_{25} & 0 & A_{27} \\ 0 & 0 & A_{33} & 0 & 0 & A_{36} & A_{37} \\ 0 & 0 & 0 & A_{44} & 0 & A_{46} & A_{47} \\ \hline A_{15}^T & A_{25}^T & 0 & 0 & A_{55} & 0 & A_{57} \\ 0 & 0 & A_{36}^T & A_{46}^T & 0 & A_{66} & A_{67} \\ \hline A_{17}^T & A_{27}^T & A_{37}^T & A_{47}^T & A_{57}^T & A_{67}^T & A_{77} \end{array}\right). \tag{4}$$

Here the separating lines represent the partition with respect to the levels of the task tree. In order to compute the factorization in parallel, we split A into the sum of four submatrices:

$$A = P_1^T \underbrace{\left(\begin{array}{cc|c} A_{11} & A_{15} & A_{17} \\ \hline A_{15}^T & A_{55}^1 & A_{57}^1 \\ \hline A_{17}^T & (A_{57}^1)^T & A_{77}^1 \end{array}\right)}_{A_1} P_1 + P_2^T \underbrace{\left(\begin{array}{cc|c} A_{22} & A_{25} & A_{27} \\ \hline A_{25}^T & A_{55}^2 & A_{57}^2 \\ \hline A_{27}^T & (A_{57}^2)^T & A_{77}^2 \end{array}\right)}_{A_2} P_2 +$$

$$P_3^T \underbrace{\left(\begin{array}{c|c|c} A_{33} & A_{36} & A_{37} \\ \hline A_{36}^T & A_{66}^3 & A_{67}^3 \\ \hline A_{37}^T & (A_{67}^3)^T & A_{77}^3 \end{array}\right)}_{A_3} P_3 + P_4^T \underbrace{\left(\begin{array}{c|c|c} A_{44} & A_{46} & A_{47} \\ \hline A_{46}^T & A_{66}^4 & A_{67}^4 \\ \hline A_{47}^T & (A_{67}^4)^T & A_{77}^4 \end{array}\right)}_{A_4} P_4, \tag{5}$$

where $P_1,\ P_2, \ldots, P_4$ denote block permutations, and $A_{55} = A_{55}^1 + A_{55}^2$, $A_{57} = A_{57}^1 + A_{57}^2$, $A_{66} = A_{66}^3 + A_{66}^4$, $A_{77} = A_{77}^1 + A_{77}^2 + A_{77}^3 + A_{77}^4$. In this paper, we have selected $A_{55}^1 = A_{55}^2$, $A_{57}^1 = A_{57}^2$, $A_{66}^3 = A_{66}^4$, $A_{67}^3 = A_{67}^4$, and $A_{77}^1 = A_{77}^2 = A_{77}^3 = A_{77}^4$, to comply with (5). In (5), the separating lines for a given A_i, represent the partitioning with respect to the path from leaf task T_i to the root.

Then, tasks $T_1,\ T_2, \ldots, T_4$ compute, respectively, multilevel ILU decompositions of $A_1,\ A_2, \ldots, A_4$ following the approach presented in Section 2. We focus next on how the decomposition of A_1 is carried out in a parallel environment. The other submatrices are treated analogously. First, step 1 from Section 2 is restricted to the A_{11} block, i.e.:

$$P = Q = \left(\begin{array}{c|c} P_{11} & 0 \\ \hline 0 & I \end{array}\right),\ D_1 = D_2 = \left(\begin{array}{c|c} D_{11} & 0 \\ \hline 0 & I \end{array}\right), \tag{6}$$

with P_{11} and D_{11} of the same row dimension as A_{11}. Next, in step 2 tiny pivots are moved to the end of the A_{11} block, instead of the end of A_1. Thus,

$$\hat{P}_1^T A_1 \hat{P}_1 = \left(\begin{array}{cc|c|c} B_{11} & F_{11} & \hat{A}_{15} & \hat{A}_{17} \\ F_{11}^T & C_{11} & \tilde{A}_{15} & \tilde{A}_{17} \\ \hline \hat{A}_{15}^T & \tilde{A}_{15}^T & A_{55}^1 & A_{57}^1 \\ \hline \hat{A}_{17}^T & \tilde{A}_{17}^T & (A_{57}^1)^T & A_{77}^1 \end{array}\right), \tag{7}$$

and we obtain the following approximate partial factorization:

$$\hat{P}_1^T A_1 \hat{P}_1 \approx \left(\begin{array}{cc|c|c} \hat{U}_{11}^T & 0 & 0 & 0 \\ \tilde{U}_{11}^T & I & 0 & 0 \\ \hline U_{15}^T & 0 & I & 0 \\ \hline U_{17}^T & 0 & 0 & I \end{array}\right) \left(\begin{array}{cc|c|c} D_{11} & 0 & 0 & 0 \\ 0 & S_{11} & S_{15} & S_{17} \\ \hline 0 & S_{15}^T & S_{55}^1 & S_{57}^1 \\ \hline 0 & S_{17}^T & (S_{57}^1)^T & S_{77}^1 \end{array}\right) \left(\begin{array}{cc|c|c} \hat{U}_{11} & \tilde{U}_{11} & U_{15} & U_{17} \\ 0 & I & 0 & 0 \\ \hline 0 & 0 & I & 0 \\ \hline 0 & 0 & 0 & I \end{array}\right). \tag{8}$$

The multilevel approach from Section 2 is repeated on the remaining Schur complement $(S_{ij})_{i,j=1,5,7}$, until either S_{11} is empty or when trying to decompose S_{11}, the number of rows factorized is small with respect to the size of S_{11}.

After computing the multilevel ILU decompositions of $A_1,\ A_2, \ldots, A_4$, we define how the parallel algorithm proceeds with the next level inside the task tree. To do this, we form matrices S_5 and S_6, which are the input matrices for T_5 and T_6, respectively. Now, we show how S_5 is built from the Schur complements computed by T_1 and T_2:

$$S_5 = \underbrace{\left(\begin{array}{cc|c|c} S_{11} & 0 & S_{15} & S_{17} \\ 0 & 0 & 0 & 0 \\ \hline S_{15}^T & 0 & S_{55}^1 & S_{57}^1 \\ \hline S_{17}^T & 0 & (S_{57}^1)^T & S_{77}^1 \end{array}\right)}_{S_1} + \underbrace{\left(\begin{array}{cc|c|c} 0 & 0 & 0 & 0 \\ 0 & S_{22} & S_{25} & S_{27} \\ \hline 0 & S_{25}^T & S_{55}^2 & S_{57}^2 \\ \hline 0 & S_{27}^T & (S_{57}^2)^T & S_{77}^2 \end{array}\right)}_{S_2}. \tag{9}$$

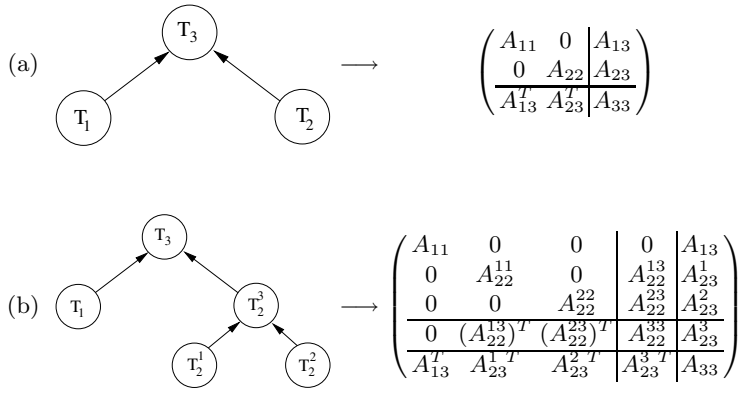


Fig. 3. Task tree height vs. number of artificial structural levels. (a) A task tree of height one introduces two levels. (b) If task T_2 of task tree (a) is split further into two leaves, the resulting task tree introduces one additional level.

Equation (9) is the key to understand how task interaction works: once T_1 and T_2 are completed, so that their corresponding Schur complements S_1 and S_2 are available, the corresponding processes in charge of T_1 and T_2 send S_1 and S_2 to the process in charge of T_5, which then computes $S_5 = S_1 + S_2$. Tasks T_3, T_4 and T_6 operate in the same way with S_3, S_4 and S_6. Then, T_5 and T_6 compute a partial multilevel ILU decomposition of S_5 and S_6, as $T_1, T_2, \ldots, T_4$ did with submatrices $A_1, A_2, \ldots, A_4$. This recursive parallel process is completed when the root task T_7 fully factorizes $S_{77}^{1,2,3,4}$, which is computed by the interaction of T_5, T_6 and T_7 once the second level of the tree is completed.

This process yields a template for the computation of multilevel preconditioners based on partial approximations such as, e.g., [13]. The procedure is easily generalized for task trees of height larger than 2, and also for non-complete[4] task trees. Finally, from the mathematical point of view, the parallel approach introduces artificial structure levels to compute the preconditioner in parallel, which may not be necessary in the sequential algorithm; see Figure 3. Therefore, the computations as well as the preconditioners computed by the sequential and the parallel algorithms are, in general, different. The same happens when computing parallel preconditioners with task trees of different heights, as we will see in Section 4.

3.2 Quality of the Task Trees

The performance of the parallel algorithm directly depends on properties of the task tree such as its shape or the computational costs associated with its tasks. Assuming the number of processors is a power of two, a desirable task tree should present the following properties:

[4] In this context, a complete task tree is one in which all leaves present the same height.

1. It should be a complete binary task tree of height $log_2(p)$, where p is the number of processors, as this enables enough concurrency for p processors (a task tree of this shape has p leaf tasks).
2. Leaf tasks should concentrate the computational cost of the whole process: in a complete binary task tree the number of processors which can operate in parallel is divided by two when we move one level higher in the tree.
3. Leaf tasks should have similar costs as, otherwise, a static mapping of tasks to processors can lead to an unbalanced distribution of the computational load.

In practice, those three properties are difficult to satisfy simultaneously. For the first two properties, remind that the task tree is constructed from the elimination tree, whose shape strongly depends on the selected coefficient matrix ordering. Thus, given an elimination tree, there is no guarantee that the desired number of independent subtrees (p) will be obtained at a certain level ($log_2(p)$). Besides, there is no guarantee either that the independent subtrees which are identified will not concentrate the bulk of the computational cost; such case happens, e.g., when the number of nodes of the independent subtrees is relatively small compared with the number of nodes of the whole elimination tree.

In order to address these two difficulties, we employ the *MultiLevel Nested Dissection* (MLND) sparse matrix ordering algorithm provided by the METIS library[5] [9]. MLND produces fill-reducing orderings with balanced elimination trees and a high degree of concurrency. Furthermore, intermediate tasks obtained as a by-product of MLND orderings correspond to node graph separators. MLND has been found to produce very small node separators, so that independent subtrees are likely to concentrate most of the nodes of the elimination tree.

MLND helps with the first two items, but we still have to address the difficulties associated with the third property: the computational costs of tasks are unknown before execution and they can be heterogenous. The cost of a task depends on many factors such as, e.g., the size of the associated block, its density and sparsity pattern, the growth of the inverse factors, etc. Unfortunately, many of these are unknown until the approximation has been computed. Therefore, it can be interesting to obtain more leaf tasks than processors and use dynamic scheduling to deal with tasks irregularity as, assuming that leaves concentrate the major part of the computation, load unbalance in the computation of leaf tasks is likely to become the main source of overhead in the parallel algorithm.

The leaf tasks which present a "high" relative cost are a potential source of load unbalance. Our approach focuses on these tasks, which are further split into finer-grain tasks. As we mentioned before, these costs are unknown a priori, so that we must use an heuristic which yields an estimation of the cost of a leaf task. Currently, our heuristic is based on the number of nonzeros of the submatrix being approximated. For example, the heuristic of the relative cost of task T_1 (see Figure 2) is $h_1 = \frac{nnz(A_1)}{nnz(A)}$, where $nnz(M)$ denotes the number of nonzero elements of the matrix M. Moreover, we must define the criterion to

[5] `http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`

decide whether to split a given leaf task. Currently, we split those leaves which satisfy $h_i > \frac{1}{f}$, where f is a user-defined threshold for the splitting mechanism. Thus, larger values of f yield a higher task tree. In this paper, we have selected f to be a modest multiple of the number of processors p. Roughly, if we select $f = kp$, we get at least k tasks per processor. For example, assuming a complete task tree of height $log_2(p)$ and $k = 1$, where $h_1 \approx h_2 \ldots \approx h_p$, we obtain p leaves. However, if some leaves at level $log_2(p)$ have a high value of h_i, then some branches are split more than others, yielding a non-complete task tree with more leaves than processors.

3.3 Implementation Details

We now discuss some aspects related with the OpenMP implementation. The task tree is constructed sequentially, prior to the parallel computation of the preconditioner. Then, the computation of the preconditioner begins, and tasks are executed by threads following a dynamic scheduling strategy, which maps tasks to threads on execution time. In general, an optimal scheduling executes *ready* tasks (i.e., those with their dependencies fulfilled) with highest computational cost as soon as possible. Our scheduling strategy applies this requirement only to the leaves of the task tree, as those concentrate the bulk of the computation. Concretely, it always gives leaves higher priority over inner tasks; among leaves, it schedules those with higher computational time first. The priority between inner tasks depends on our implementation, which we discuss next.

The key of our implementation is a *ready queue* which contains those tasks with their dependencies resolved. Tasks are dequeued for execution from the head, and new ready tasks are enqueued at the tail of this structure. Leaf tasks are initially enqueued in order of priority following our heuristic approach; see Figure 4 (a). The execution of tasks is scheduled dynamically: whenever threads become idle, they monitor the queue for pending tasks. When a thread completes execution of a task, the task dependent on it is examined, and if its dependencies have been resolved, then it is enqueued at the ready queue by this thread. Figure 4 (b) illustrates an example scenario of how this mechanism works with $p = 2$

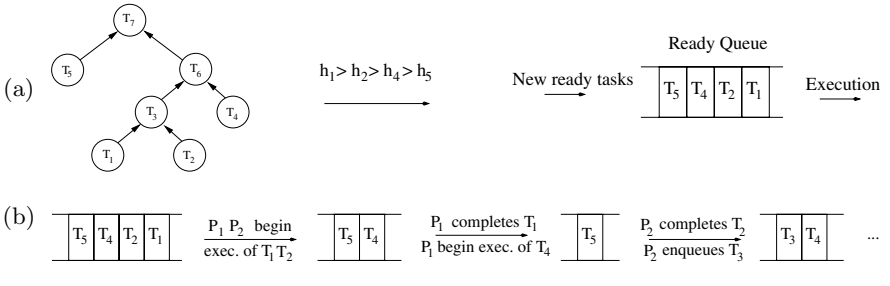


Fig. 4. Dynamic scheduling implementation. (a) The initial state of the ready queue. (b) An example scenario of how dynamic scheduling works with $p = 2$. Note that in this situation there has been a heuristic missestimation: T_1 is completed before T_2.

and the task tree of Figure 4 (a). As can be observed, when thread 2 (i.e., P_2) completes the execution of T_2, dependencies of T_3 are resolved and T_3 becomes ready. Idle threads continue to dequeue tasks until all tasks have been executed. Similar mechanisms have been proposed for irregular codes as part of the Cilk project [10] and for dense linear algebra codes in the FLAME project [6].

4 Experimental Results

All experiments in this section were obtained on a SGI Altix 350 CC-NUMA multiprocessor consisting of 16 Intel Itanium2@1.5 GHz processors sharing 32 GBytes of RAM via a SGI NUMAlink interconnect. IEEE double-precision arithmetic was employed in all experiments, and one thread was binded per processor. In both the serial and parallel algorithms we used ILUPACK default values for the condition estimator (κ=100), as well as default the drop tolerances for the L factor (ϵ_1=10^{-2}) and the Schur complement (ϵ_2=10^{-2}).

Table 1 characterizes the benchmark matrices from the UF sparse matrix collection[6] employed in the evaluation. As shown there, they arise from different application areas. Table 2 reports the results obtained from the execution of the serial algorithm in ILUPACK: serial execution time and fill-in factor (ratio between the number of nonzero elements in the triangular factors produced by the algorithm and the coefficient matrix). In the multilevel configuration, the input matrix for the first level has been pre-ordered with MLND, while the input matrices for the rest of the levels are not pre-ordered.

Table 1. Benchmark matrices selected to test the parallel multilevel ILU

Code	Group/Name	Problem Domain	Rows/Cols.	Nonzeros
M1	*GHS_psdef/bmwcra_1*	Structural Problem	148770	10641602
M2	*Wissgott/parabolic_fem*	Computational Fluid Dynamics	525825	3674625
M3	*Schmid/thermal2*	Thermal Problem	1228045	8580313
M4	*AMD/G3_circuit*	Circuit Simulation Problem	1585478	7660826

Table 2. Results from the execution of the serial algorithm in ILUPACK

Code	M1	M2	M3	M4
Time (secs.)	98.6	9.63	22.4	32.1
Fill-in factor	7.2	4.4	4.3	5.0

For the parallel algorithm, the multilevel configuration of the sequential algorithm is kept. For simplicity, we only present results for $f = p, 2p, 4p$, with $p = 2, 4, 8, 16$ processors, but note that our implementation also allows values of

[6] `http://www.cise.ufl.edu/research/sparse/matrices`

f and p which are not power of two, e.g., $p = 12$ and $f = 3p$, and values of f which are not a multiple of p. Therefore, we use six different values of f, resulting in as many different preconditioners. Some of these are computed varying the number of processors; thus, e.g., $f = 8$ is used for $p = 2, 4, 8$ processors. Table 3 reports the number of leaves in the task trees and the fill-in factor for the corresponding preconditioner obtained by various choices of f. The fill-in factors are similar to those attained by the serial algorithm in ILUPACK, except for matrix M_1, for which the fill-in tends to be more reduced. The height of the tree depends on the value of f. As stated in Section 3.1, the computed preconditioners may differ when employing task trees of different heights. Therefore, the fill-in may change.

To give a rough idea of how our approach will perform, we simulate how dynamic scheduling would work in case of $h_i = c_i$, where c_i is the relative computational cost of a given task T_i. For example, assuming the task tree of Figure 4 (a), with $h_1 = 0.39$, $h_2 = 0.29$, $h_4 = 0.2$ and $h_5 = 0.1$, and $p = 2$, then the leaf tasks scheduling events are simulated to happen in the following order: T_1 scheduled to P_1, T_2 scheduled to P_2, T_4 scheduled to P_2, and T_5 scheduled to P_1. If we define $P_1 = h_1 + h_5$ and $P_2 = h_2 + h_4$, we can compute the variation coefficient $c_h = \sigma_h / \bar{h}$ with σ_h the standard deviation and $\bar{h}$ the average of these values (i.e., P_1 and P_2). Table 3 reports the value of c_h (expressed as a percentage) of each combination for all four matrices M1-M4, fraction, and number of processors. A ratio close to 0% implies an accurate heuristic balancing (e.g., M2/$f = 16/p = 16$), while a ratio significantly large (e.g., M1/$f = 16/p = 16$) indicates heuristic unbalance. As can be observed in Table 3, it is a good strategy to employ the same task tree while reducing the number of processors. For example, for matrix M1/$f = 16$, c_h is reduced by factor of 73% when we use $p = 4$ instead of $p = 16$ processors. Therefore, in general, a larger number of leaves per processor increases heuristic balancing. For matrix M2 this difference is not observed. This is due to the structure of the graph underlying M2, which allows MLND to produce high balanced elimination trees. This can be observed from the fact that, for $f = 2, 4, 8, 16$, we obtain complete binary task trees with f leaves.

Table 4 reports the execution time and the speed-up of the parallel algorithm. The speed-up is computed with respect to the parallel algorithm executed using the same task tree on a single processor. A comparison with the serial algorithm in ILUPACK offers superlinear speed-ups in many cases (see execution times in Table 1) and has little meaning here. Table 4 also reports the variation coefficient c_t (expressed as a percentage), which takes into consideration measured data from the execution instead of estimations, as c_h do. A ratio close to 0% indicates a balanced distribution of the workload (e.g., M2/$f = 16/p = 16$), while a ratio significantly large (e.g., M1/$f = 16/p = 16$) indicates an unbalanced distribution of the computational load.

For matrices M2, M3, M4, and $p = 2, 4, 8, 16$, it was enough to select $f = p$ in order to get remarkable speed-ups (close to linear). This situation was well predicted by our current heuristic, as can be observed by comparing the values of c_h and c_t in Tables 3- 4. However, for matrix M1 it is not enough to select

Table 3. Number of leaf tasks of the tasks trees constructed, fill-in generated for each selected value of f and variation coefficient for each combination of f and p

M1					M2					M3					M4				
f	Leaves	Fill-in	p	c_h	f	Leaves	Fill-in	p	c_h	f	Leaves	Fill-in	p	c_h	f	Leaves	Fill-in	p	c_h
2	2	6.3	2	1	2	2	4.4	2	0	2	4	4.3	2	1	2	5	5.0	2	0
4	6	4.6	2	1	4	4	4.4	2	0	4	7	4.3	2	0	4	11	5.0	2	0
			4	4				4	0				4	1				4	0
8	10	4.2	2	0	8	8	4.4	2	0	8	12	4.3	2	1	8	19	5.0	2	1
			4	8				4	0				4	1				4	1
			8	9				8	0				8	2				8	2
16	20	3.4	4	3	16	16	4.5	4	0	16	22	4.3	4	0	16	38	5.0	4	0
			8	9				8	0				8	2				8	1
			16	11				16	0				16	2				16	2
32	36	2.9	8	5	32	34	4.5	8	2	32	43	4.4	8	1	32	74	5.0	8	1
			16	9				16	2				16	2				16	1
64	66	2.7	16	5	64	70	4.4	16	3	64	83	4.4	16	1	64	158	5.0	16	0

$f = p$; see, e.g., what happens for $f = 8/p = 8$ and $f = 16/p = 16$, where the speed-up are 4.74 and 9.26 respectively. This situation was not well predicted by our heuristic, which makes too optimistic predictions in these situations. A closer inspection of the task tree for $f = 16$, reveals 20 highly irregular leaf tasks to be mapped to $p = 16$ processors. In this case, dynamic mapping is useless, and hence the computational load is unbalanced. However, using this task tree with $p = 8$ and $p = 4$ leads to better results: when obtaining more leaves per processor, dynamic mapping is able to balance the computational load. Further splitting the task tree led to better performance in many situations, as happened, e.g., for M1/$p = 8$, where the speed-up was increased from 4.74 ($f = 8$) to 7.54 ($f = 32$). Finally, the relationship between c_t and the speed-up, confirms that the computational cost is concentrated on the leaves, and hence load balancing in the computation of these tasks led to high parallel performance.

We also traced the execution of our parallel algorithm with the KOJAK and VAMPIR performance analysis tools. Concretely, we focus on the influence of the potential sources of overhead in our parallel algorithm as idle threads, synchronization, access to shared resources on the SMM, CC-NUMA data locality, etc. We observed that idle threads are the main source of overhead, while the influence of other factors is negligible. For example, for $p = 16$ and matrix M1, the mean time to access the shared queue used for the dynamic scheduling implementation is approximately 40 μsecs. Therefore, as we employ large-grain parallelism (i.e., f a modest multiple of p), this overall synchronization overhead does not hurt performance.

In some situations dynamic mapping could have further reduced idle threads with a better heuristic estimation. To reduce idle threads it is mandatory to schedule those tasks which are in the critical path (that is, those leaf tasks with

Table 4. Performance of the parallel multilevel ILU

M1					M2					M3					M4				
f	p	Time(sec.)	Speed-Up	c_t	f	p	Time(sec.)	Speed-Up	c_t	f	p	Time(sec.)	Speed-Up	c_t	f	p	Time(sec.)	Speed-Up	c_t
2	2	43.7	1.92	1	2	2	4.88	1.97	0	2	2	11.1	1.97	0	2	2	16.1	1.98	0
4	2	28.6	1.96	2	4	2	4.77	1.99	1	4	2	11.0	1.98	1	4	2	16.1	1.98	1
	4	15.9	3.52	11		4	2.41	3.93	1		4	5.58	3.91	1		4	8.20	3.89	1
8	2	28.1	1.92	4	8	2	4.70	1.99	0	8	2	11.1	1.96	2	8	2	16.2	1.96	2
	4	15.2	3.57	10		4	2.37	3.95	0		4	5.59	3.89	2		4	8.22	3.87	2
	8	11.4	4.74	35		8	1.23	7.63	2		8	2.89	7.54	2		8	4.20	7.57	2
16	4	13.2	3.81	4	16	4	2.33	3.98	0	16	4	5.53	3.92	2	16	4	8.09	3.90	2
	8	8.55	5.91	18		8	1.20	7.77	1		8	2.82	7.67	2		8	4.23	7.46	3
	16	5.45	9.26	37		16	0.65	14.3	3		16	1.46	14.8	2		16	2.26	14.0	3
32	8	6.00	7.54	2	32	8	1.20	7.61	2	32	8	2.77	7.76	1	32	8	4.14	7.60	2
	16	3.44	13.1	7		16	0.65	14.0	3		16	1.45	14.8	2		16	2.21	14.2	4
64	16	3.61	13.0	7	64	16	0.62	14.6	3	64	16	1.42	14.9	2	64	16	2.16	14.6	2

higher cost) as soon as possible. As mentioned in Section 3.3, we priorize the execution of leaves with higher heuristic by inserting them first in the queue. If some leaves costs are missestimated dynamic mapping can not comply with this requirement. Figure 5 shows a capture of the global timeline view of VAMPIR when displaying a trace of our parallel algorithm with M1/$f = 16/p = 8$. For each thread (i.e., each horizontal stripe), the display shows the different states and their change over execution time along a horizontal time axis. Concretely, black bars represent time intervals where threads are executing tasks, while white bars represent other thread states, such as idle or synchronization states. As can be observed, the two boxed leaf tasks present a high relative cost, and have been scheduled after other leaves with less computational cost, due to heuristic missestimation. Here, delays due to idle threads are further reduced by splitting the task tree: thus, the efficiency attained for M1/$f = 16/p = 4$ is 95%.

Finally, we employed our own implementation [2] of the PCG solver to evaluate and compare the numerical quality of our parallel preconditioners and those computed by ILUPACK serial routines. The preconditioned iteration was stopped when either the iteration count exceeded 1000 iterations or the iterates satisfied the stopping criterion described in [11]. Table 5 shows the number of iteration steps required by the PCG to solve the preconditioned linear systems considered, as a function of the f parameter. The values at column $f = 1$ correspond to ILUPACK preconditioning serial routines. For matrix M1, we changed the default value for the condition estimator from κ=100 to κ=5 because the PCG solver did not converge within the maximum prescribed number of iteration steps. The preliminary results shown at Table 5 confirm that the numerical quality of the computed parallel preconditioners mildly depends on the task tree. This is currently under investigation.

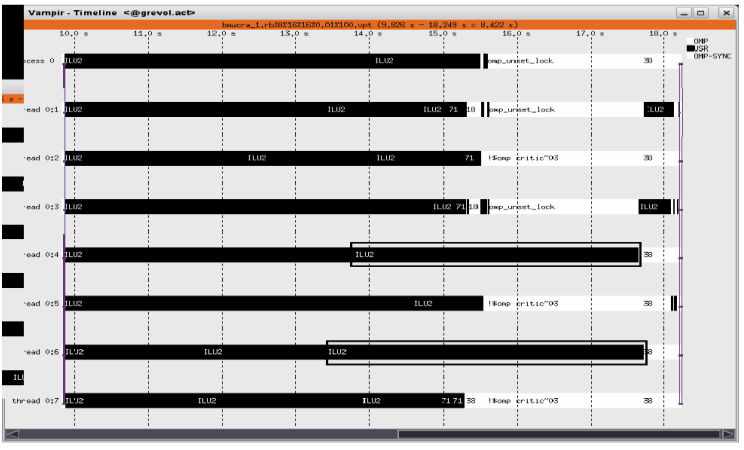


Fig. 5. VAMPIR global timeline view for M1/$f = 16/p = 8$. The two boxed leaf tasks present a high relative cost, and have been scheduled after other leaves with less computational cost.

Table 5. Iteration count for the PCG with our parallel multilevel ILU preconditioners

Code\f	1	2	4	8	16	32	64
M1	780	836	839	855	842	835	828
M2	90	93	92	95	104	107	112
M3	138	147	156	162	165	170	185
M4	69	75	77	80	79	79	85

5 Conclusions and Future Work

We have presented in detail the design foundations of a parallel multilevel preconditioner for the iterative solution of sparse linear systems using Krylov subspace methods. Our OpenMP implementation internally employs the serial routines in ILUPACK. MLND ordering, task splitting, and dynamic scheduling of tasks are used to enhance the degree of parallelism of the computational procedure. The use of large-grain decompositions ($f = p$) led to poor performance in some situations. This can be solved by obtaining finer-grain decompositions ($f = 4p$) combined with dynamic scheduling to deal with tasks irregularity. Remarkable performance has been reported on a CC-NUMA platform with 16 Itanium2 processors and we have gained more insights on the performance attained with the performance analysis tools employed.

Future work includes:

- To compare and contrast the numerical properties of the preconditioner in ILUPACK and our parallel preconditioner.
- To develop parallel algorithms to apply the preconditioner to the system, and to solve the system iteratively.

– To develop parallel preconditioners for non-SPD linear systems.
– To analyze alternative heuristics.

Acknowledgments

This research has been supported by the CICYT project TIN2005-09037-C02-02 and FEDER, and the DAAD D-07-13360/*Acciones Integradas Hispano-Alemanas* programme HA2007-0071. Alberto F. Martín was also supported by a research fellowship from the *Generalitat Valenciana* (BFPI/06). This work was partially carried out during a visit of Alberto F. Martín to the TU-Braunschweig supported by the programme *Plan 2007 de Promoción de la Investigación* of the Universidad Jaime I.

References

1. Aliaga, J.I., Bollhoefer, M., Martín, A.F., Quintana-Ortí, E.S.: Parallelization of Multilevel Preconditioners Constructed from Inverse-Based ILUs on Shared-Memory Multiprocessors. In: Bischof, C., et al. (eds.) Parallel Computing: Architectures, Algorithms and Applications, pp. 287–294 (2007)
2. Aliaga, J.I., Bollhoefer, M., Martín, A.F., Quintana-Ortí, E.S.: Scheduling Strategies for Parallel Sparse Backward/Forward Substitution PARA 2008, Trondheim (2008) (under revision)
3. Bollhoefer, M.: A Robust ILU Based on Monitoring the Growth of the Inverse Factors. Linear Algebra Appl. 338(1-3), 201–218 (2001)
4. Bollhoefer, M.: A robust and efficient ILU that incorporates the growth of the inverse triangular factors. SIAM J. Sci. Comput. 25(1), 86–103 (2003)
5. Bollhoefer, M., Saad, Y.: Multilevel preconditioners constructed from inverse–based ILUs. SIAM J. Sci. Comput. 25(5), 1627–1650 (2006)
6. Chan, E., Quintana-Ortí, E.S., Quintana-Ortí, G., van de Geijn, R.: SuperMatrix out-of-order scheduling of matrix operations for SMP and multi-core architectures. In: Proceed. 19th ACM SPAA 2007, pp. 116–125 (2007)
7. Davis, T.: Direct methods for sparse linear systems. SIAM Publications, Philadelphia (2006)
8. Demmel, W., Gilbert, J.R., Li, X.S.: An Asynchronous Parallel Supernodal Algorithm for Sparse Gaussian Elimination. SIAM J. Matrix. Anal. Appl. 20(4), 915–952 (1999)
9. Karypis, G., Kumar, V.: A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM J. Sci. Comput. 20(1), 359–392 (1998)
10. Leiserson, C., Plaat, A.: Programming parallel applications in Cilk. SIAM News, SINEWS (1998)
11. Strakos, Z., Tichy, P.: Error Estimation in Preconditioned Conjugate Gradients. BIT Numerical Mathematics 45, 789–817 (2005)
12. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM Publications, Philadelphia (2003)
13. Saad, Y.: Multilevel ILU with reorderings for diagonal dominance. SIAM J. Sci. Comput. 27(3), 1032–1057 (2005)
14. Schenk, O., Bollhöfer, M., Römer, R.A.: On Large Scale Diagonalization Techniques for the Anderson Model of Localization. SIAM Review 50(1), 91–112 (2008)

On the I/O Volume in Out-of-Core Multifrontal Methods with a Flexible Allocation Scheme*

Emmanuel Agullo[1,3,6], Abdou Guermouche[4,5], and Jean-Yves L'Excellent[2,3,6]

[1] École Normale Supérieure de Lyon
[2] Institut National de Recherche en Informatique et en Automatique (Inria)
[3] Laboratoire de l'Informatique du Parallélisme (UMR CNRS-ENS Lyon-INRIA-UCBL), ÉNS Lyon, 46 allée d'Italie, 69364 Lyon cedex 07, France
[4] Laboratoire Bordelais de Recheche en Informatique (UMR 5800) - 351, cours de la Liberation F-33405 Talence cedex, France
[5] Université de Bordeaux 1
[6] Université de Lyon
Emmanuel.Agullo@ens-lyon.fr,
Abdou.Guermouche@labri.fr,
Jean-Yves.L.Excellent@ens-lyon.fr

Abstract. Sparse direct solvers, and in particular multifrontal methods, are widely used in various simulation problems. Because of their large memory requirements, the use of *out-of-core* solvers is compulsory for large-scale problems, where disks are used to extend the core memory. This study is at the junction of two previous independent works: it extends the problem of the minimization of the volume of *I/O* [4] in the multifrontal method to the more general *flexible parent allocation* algorithm [8]. We explain how to compute the *I/O* volume with respect to this scheme and propose an efficient greedy heuristic which significantly reduces the *I/O* volume on real-life problems in this new context.

1 Introduction

We are interested in the direct solution of systems of equations of the form $Ax = b$, where A is a large sparse matrix. In direct methods, because the storage requirements are large compared to the initial matrix A, out-of-core approaches may become necessary. In such cases, left-looking [13,14] and multifrontal [1,12] methods are the two most widely used approaches. One drawback of multifrontal methods comes from large dense matrices that give a lower bound on the minimum core memory requirements. However, those dense matrices may fit in memory, or they can be treated with an out-of-core process. Apart from these dense matrices, the out-of-core multifrontal method follows a write-once/read-once scheme, which makes it interesting when one is interested in limiting the volume of *I/O*. For matrices A with a symmetric structure (or in approaches like [6] when the structure of A is unsymmetric), the dependency graph of the multifrontal approach is given by a tree, processed from leaves to root. The tree structure results from the sparsity of the matrix and from the order in which the variables

* Partially supported by ANR project SOLSTICE, ANR-06-CIS6-010.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 328–335, 2008.

of the sparse matrix are eliminated; different branches of the tree may be processed independently. To each node of the tree is associated a so called *frontal matrix*, which is a dense matrix divided into two parts (see Figure 1, left): (i) a *fully summed block*, that will be factorized, and a non-fully summed block that cannot be factorized yet but will be updated and used later at the parent node, after it has been replaced by a *Schur complement*, or *contribution block*. To be more precise, the following tasks are performed at each node of the tree:

(i) allocation of the *frontal matrix* in memory;
(ii) assembly of data (*contribution blocks*) coming from the child nodes into that frontal matrix;
(iii) partial factorization of the fully summed part of the frontal matrix, and update of the rest.

After step (iii), the fully summed part of the frontal matrix has been modified and contains *factors*, that will only be reused at the solution stage, whereas the non fully summed part contains the Schur complement, that will contribute to the frontal matrix from the parent node (see Figure 1, right). Because factors are not re-accessed during the multifrontal factorization, they can be written to disk directly, freeing some storage. Then remains the temporary storage associated to the contribution blocks waiting to be assembled and to the current frontal matrix. In the classical multifrontal scheme, the frontal matrix of a parent is allocated (and then factored) only after all children have been processed. We call this approach *terminal allocation*. Assuming that a postorder of the tree is used, contribution blocks can then be managed thanks to a stack mechanism. Furthermore, the order in which the children are processed (or tree traversal) has a strong impact on both the storage requirement for the stack and the volume of *I/O*, should this stack be processed *out-of-core*.

A more extensive description of the multifrontal approach can be found in, for example, [7,11]. In general a large workarray is pre-allocated, in order to store the current frontal matrix and the contribution blocks. Allowing the frontal matrix of the parent to overlap with the contribution block of the last child is possible, and significantly reduces the overall storage requirement. Considering a so-called *family* composed of a parent node, with a frontal matrix of size m, and its set of n children that produce contribution

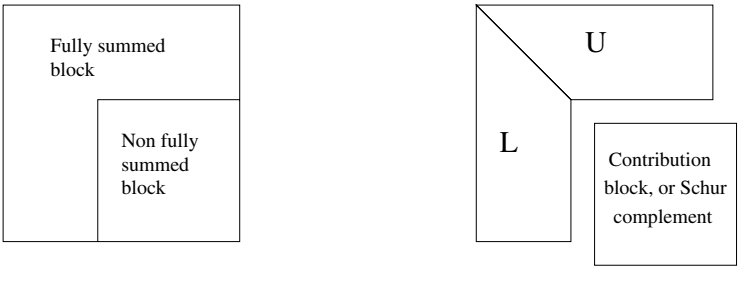


Fig. 1. Frontal matrix at a node of the tree before (left) and after (right) the partial factorization of step (iii) in the unsymmetric case (LU factorization)

blocks of size $cb_i,\ i = 1, \ldots, n$, [10] shows that the storage requirement to process the tree rooted at the parent is

$$S^{terminal} = \max\left(\max_{j=1,n}(S_j^{terminal} + \sum_{k=1}^{j-1} cb_k), m + \sum_{k=1}^{n-1} cb_k\right) \quad (1)$$

(where $S_j^{terminal}$ is recursively the storage for the subtree rooted at child j) and can be minimized by sorting the children in decreasing order of $\max(S_j^{terminal}, m) - cb_j$. By applying this formula and this ordering at each level of the tree, we obtain the volume of *I/O* for the complete tree, together with the tree traversal. Starting from (1), we have shown in [4] that for a given amount of available memory, M_0, the volume of *I/O* (=volume written=volume read) associated to the temporary storage of the multifrontal method is

$$V^{terminal} = \max\left(0, \max(\max_{j=1,n}(\min(S_j^{terminal}, M_0) + \sum_{k=1}^{j-1} cb_k), m + \sum_{k=1}^{n-1} cb_k) - M_0\right) + \sum_{j=1}^{n} V_j^{terminal} \quad (2)$$

which is minimized by sorting the children in decreasing order of

$$\max(\min(S_j^{terminal}, M_0), m) - cb_j$$

at each level of the tree. This gives an optimal tree traversal which is different from the one from [10]: minimizing the *I/O* volume is different from minimizing the overall storage requirement.

2 Flexible Parent Allocation

With the *terminal allocation* scheme, steps (i), (ii) and (iii) for a parent node are only performed when all children have been processed. However, the main constraint is that the partial factorization (step (iii) above) at the parent level must be performed after the assembly step (ii) has been performed for all child contributions. Thus, the allocation of the parent node (step (i)), and the assembly of the contributions of some children can be performed (and the corresponding contribution block freed) without waiting that all children have been processed. This flexibility has been exploited by [8] to further reduce the storage requirement for temporary data. Assume that the parent node is allocated after p children have been processed, and that the memory for the p^{th} child overlaps with the memory for the parent. The storage required for a parent in this *flexible* scheme is then given by:

$$S^{flex} = \max\left(\max_{j=1,\boxed{p}}(S_j^{flex} + \sum_{k=1}^{j-1} cb_k), m + \sum_{k=1}^{\boxed{p}-1} cb_k, m + \max_{j=p+1,n} S_j^{flex}\right) \quad (3)$$

When the parent is allocated, all the contributions from its factored children are assembled and discarded. From that point on, each child that is factored sees its contribution block immediately assembled and its memory is released. [8] shows how to choose the point (*split point*) where the parent should be allocated and how to order the children so that the storage requirement S^{flex} is minimized.

Now if the value of S^{flex} is larger than the available memory, then disk storage must be used. In that case, rather than minimizing S^{flex}, it becomes more interesting to minimize the volume of *I/O*: this is the objective of the current paper. To limit the volume of *I/O*, minimizing S^{flex} can appear like a good heuristic. In [12], the authors have done so, adapting [8] with respect to some additional constraints imposed by their code. However, by computing the volume of *I/O* formally, we can show the limits of a memory-minimizing approach when aiming at decreasing the *I/O* volume: similarly to the terminal allocation case, minimizing the volume of *I/O* in the flexible allocation scheme is different from minimizing the storage requirement.

3 Volume of *I/O* in a Flexible Multifrontal Method

The main difference compared to (2) is that with a *flexible* allocation scheme, a child j processed after the parent allocation ($j > p$) may also generate *I/O*. Indeed, if this child cannot be processed *in-core* together with the frontal matrix of the parent, then part (or the whole) of the frontal matrix has to be written to disk in order to make room and process the child with a maximum of available memory. This possible extra-*I/O* corresponds to underbrace **(a)** of Formula (4). After that, the factor block of the frontal matrix of child j is written to disk and its contribution block is ready to be assembled into the frontal matrix of the parent; to do so, and because we cannot easily rely on a simple property to find which rows of the contribution block, if any, can be assembled into the part of the frontal matrix available in memory, we assume that this latter frontal matrix is fully re-loaded into memory (reading back from disk the part previously written). This operation may again generate *I/O*: if the contribution block of child j and the frontal matrix of its parent cannot hold together in memory, a part of cb_j has to be written to disk, then read back (panel by panel) and finally assembled. This second possible extra-*I/O* is counted in underbrace **(b)** of Formula (4). All in all, and using the storage definition from Formula (3), the volume of *I/O* required to process the subtree rooted at the parent node is given by:

$$V^{flex} = \max\left(0, \max\left(\max_{j=1,\boxed{p}}\left(\min(S_j^{flex}, M_0) + \sum_{k=1}^{j-1} cb_k\right), m + \sum_{k=1}^{\boxed{p}-1} cb_k\right) - M_0\right) + \sum_{j=1}^{n} V_j^{flex} + \underbrace{\sum_{j=p+1}^{n}\left(\max(0, m + \min(S_j^{flex}, M_0) - M_0)\right)}_{\textbf{(a)}} + \underbrace{\sum_{j=p+1}^{n}(\max(0, m + cb_j - M_0))}_{\textbf{(b)}} \quad (4)$$

Again, a recursion gives the *I/O* volume for the whole tree.

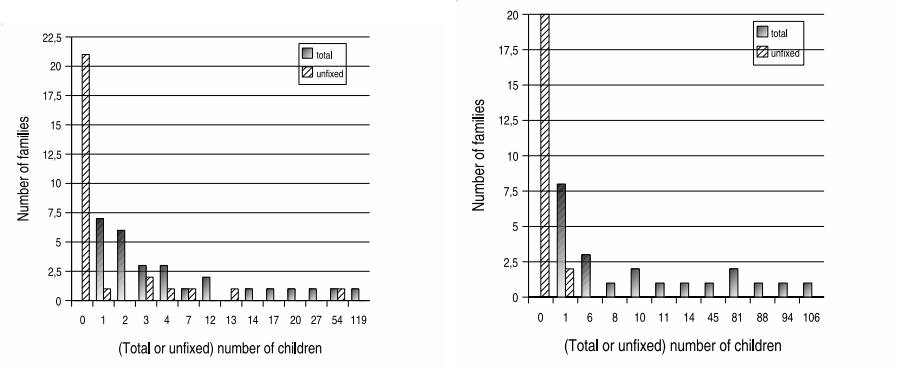
4 Minimizing the *I/O* Volume in the Flexible Multifrontal Method

With the terminal allocation scheme, the *I/O* volume (on a parent node and its n children) is minimized by sorting the children in an appropriate order. With the flexible scheme, one should *moreover* determine the appropriate *split point*, *i.e.* the best value for p. In other words, the flexible *I/O* volume is minimized when together **(i)** the children processed *before* the parent allocation are correctly separated from the ones processed *after* and **(ii)** each one of this set is processed in an appropriate order. Exploring these $n.n!$ combinations is not always conceivable since some families may have a very large number n of children (more than one hundred for instance for the GUPTA3 matrix). However, relying on [4] and Formula 2, we know that an optimal order among children processed before the parent allocation is obtained when they are sorted in decreasing order of $\max(\min(S_j^{flex}, M_0), m) - cb_j$. Moreover, the *I/O* volume on the children processed after the allocation is independent of their relative processing order. Said differently, these two remarks mean that **(ii)** is actually obvious when **(i)** is determined: we only have to determine to which set (before or after the parent allocation) each child belongs to. But this still makes an exponential (2^n) number of possibilities to explore.

Actually, the decision problem associated to this minimization problem is NP-complete. In other words, given an arbitrary target amount of *I/O* V, there is no deterministic polynomial algorithm that can consistently decide whether there exists a partition of the children inducing a volume of *I/O* lower than or equal to V (except if $P = NP$). The proof of the NP-completeness (reduction from 2-PARTITION) is out-of-scope for this paper and is provided in [3].

To further reduce the complexity, remark that if a child is such that $m + S_j^{flex} \leq M_0$, ordering this child *after* the parent allocation does not introduce any additional *I/O* (**(a)** and **(b)** are both 0 in (4)), whereas this may not be the case if it is processed before the parent allocation. Therefore, we conclude that we can place all children verifying $m + S_j^{flex} \leq M_0$ after the parent allocation. Furthermore, consider the case where $S_j^{flex} \geq M_0 - m + cb_j$ and $m + cb_j \leq M_0$. Processing this child *after* the parent allocation (see Formula (4)) leads to a volume of *I/O* either equal to m (if $S_j^{flex} >= M_0$) – which is greater than cb_j, or to $S_j^{flex} - M_0 + m$ (if $S_j^{flex} \leq M_0$) – which is also greater than cb_j. On the other hand, treating that child *first* (this may not be optimal) will lead to a maximum additional volume of *I/O* equal to cb_j. Therefore, we can conclude that we should process it *before* the parent allocation. For the same type of reasons, children verifying $S_j^{flex} \geq 2(M_0 - m)$ and $m + cb_j > M_0$ should also be processed *before* the parent allocation.

We will say that a child is *fixed* if it verifies one of the above properties: a straightforward analysis - independent of the metrics of its siblings - determines if it should be processed before or after the allocation of the parent node. Even though the number of *fixed* children can be large in practice, some matrices may have a few families with a large number of *unfixed* children, as shown in Figure 2 for two sparse problems. For instance, among the 28 families inducing *I/O* for the GUPTA3 matrix ordered with METIS when a memory of $M_0 = 684686$ reals is available, 21 families have no unfixed children (thus for them the optimum process is directly known), but one family keeps having 54 unfixed children. In such cases, heuristics are necessary and we designed one



(a) GUPTA3 matrix - METIS ordering M_0=684686 (b) TWOTONE matrix - PORD ordering M_0=7572632

Fig. 2. Distribution of the families in function of their total and unfixed number of children. After a straightforward analysis, most families have few (or no) unfixed children.

consisting in moving after the allocation the child which is responsible for the peak of storage until one move does not decrease the volume of *I/O* anymore.

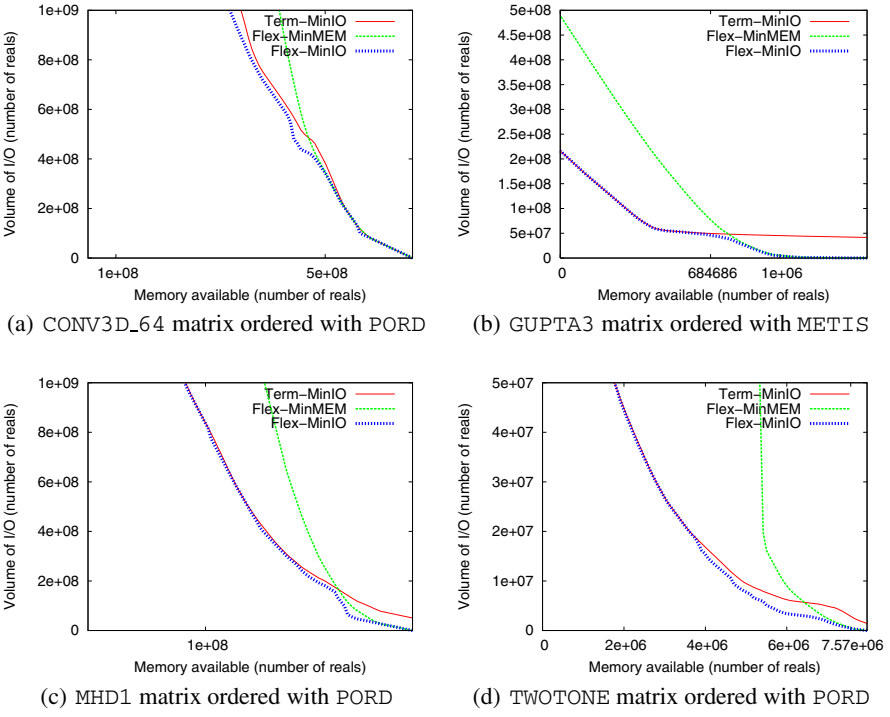
5 Experimental Results

In order to evaluate the impact of this flexible allocation scheme on the volume of *I/O*, we compare the results of our heuristic (`Flex-MinIO`) both to the terminal allocation scheme with the IO-minimizing algorithm of [4] (`Term-MinIO`) and to the flexible allocation scheme with the memory-minimizing algorithm of [8] (`Flex-MinMEM`).

The volumes of *I/O* were computed by instrumenting the analysis phase of `MUMPS` [5] which allowed us to experiment several ordering heuristics. We retained results with both `METIS` [9] and `PORD` [15]. For a given matrix, an ordering heuristic defines the order in which the variables of the matrix are eliminated and an associated task dependency graph (or tree, see Section 1). It impacts the computational complexity as well as different metrics such as the volume of *I/O*.

Table 1. Test problems. Size of factors ($nnz(L|U)$) and number of floating-point operations (Flops) were computed with PORD ordering, except the ones of `GUPTA3` for which METIS ordering was used.

Matrix	Order	nnz	Type	$nnz(L\|U)$ ($\times 10^6$)	Flops ($\times 10^9$)	Description
`CONV3D_64`	836550	12548250	UNS	4690.6	48520	Provided by CEA-CESTA; generated using AQUILON (`http://www.enscpb.fr/master/aquilon`).
`GUPTA3`	16783	4670105	SYM	10.1	6.3	Linear programming matrix (AA'), Anshul Gupta (Univ. Florida collection).
`MHD1`	485597	24233141	UNS	1169.7	8382	Unsymmetric magneto-hydrodynamic 3D problem, provided by Pierre Ramet.
`TWOTONE`	120750	1224224	UNS	30.7	39.7	AT&T,harmonic balance method, two-tone. More off-diag nz than onetone (Univ. Florida collection).



(a) CONV3D_64 matrix ordered with PORD (b) GUPTA3 matrix ordered with METIS

(c) MHD1 matrix ordered with PORD (d) TWOTONE matrix ordered with PORD

Fig. 3. I/O volume on the stack of contribution blocks as a function of the core memory available for the three heuristics with four different matrices

We have selected four test problems that we present in Table 1 and for which we have observed significant gains.

Figure 3 shows the evolution of the corresponding volume of *I/O* with the available memory on the target machine. When a large amount of memory is available (right part of the graphs), the flexible allocation schemes (both `Flex-MinMEM` and `Flex-MinIO`) induce a small amount of *I/O* compared to the terminal allocation scheme (`Term-MinIO`). Indeed, with such an amount of memory, many children can be processed after the allocation of their parent without inducing any *I/O* (or inducing a small amount of *I/O*): the possible extra-*I/O*s corresponding to underbraces **(a)** and **(b)** of Formula (4) are actually equal (or almost equal) to zero for those children.

When the amount of available memory is small (left part of the graphs), the memory-minimizing algorithm (`Flex-MinMEM`) induces a very large amount of *I/O* compared to the *I/O*-minimization algorithms (both `Flex-MinIO` and `Term-MinIO`). Indeed, processing a child after the parent allocation may then induce a very large amount of *I/O* (M_0 is small in underbraces **(a)** and **(b)** of Formula (4)) but memory-minimization algorithms do not take into account the amount of available memory to choose the *split point*.

Finally, when the amount of available memory is intermediate, the heuristic we have proposed (`Flex-MinIO`) induces less *I/O* than both other approaches. Indeed, accord-

ing to the memory, not only does the heuristic use a flexible allocation scheme on the families for which it is profitable, but it can also adapt the number of children to be processed after the parent allocation.

6 Conclusion

The algorithms presented in this paper should allow to improve the new generation of serial [12] *out-of-core* multifrontal codes, based on the flexible allocation scheme, as well as the serial parts of the parallel ones [2]. Implementation issues of the method are further discussed in [3].

References

1. The BCSLIB Mathematical/Statistical Library, `http://www.boeing.com/phantom/bcslib/`
2. Agullo, E., Guermouche, A., L'Excellent, J.-Y.: Towards a parallel out-of-core multifrontal solver: Preliminary study. Research report 6120, INRIA. Also appeared as LIP report RR2007-06 (February 2007)
3. Agullo, E.: On the out-of-core factorization of large sparse matrices. Phd thesis, Ecole Normale Supérieure de Lyon (October 2008) (submitted)
4. Agullo, E., Guermouche, A., L'Excellent, J.-Y.: On reducing the I/O volume in a sparse out-of-core solver. In: Aluru, S., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2007. LNCS, vol. 4873, pp. 47–58. Springer, Heidelberg (2007)
5. Amestoy, P.R., Guermouche, A., L'Excellent, J.-Y., Pralet, S.: Hybrid scheduling for the parallel solution of linear systems. Parallel Computing 32(2), 136–156 (2006)
6. Amestoy, P.R., Puglisi, C.: An unsymmetrized multifrontal LU factorization. SIAM Journal on Matrix Analysis and Applications 24, 553–569 (2002)
7. Duff, I.S., Reid, J.K.: The multifrontal solution of indefinite sparse symmetric linear systems. ACM Transactions on Mathematical Software 9, 302–325 (1983)
8. Guermouche, A., L'Excellent, J.-Y.: Constructing memory-minimizing schedules for multifrontal methods. ACM Transactions on Mathematical Software 32(1), 17–32 (2006)
9. Karypis, G., Kumar, V.: MeTiS – A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0. University of Minnesota (September 1998)
10. Liu, J.W.H.: On the storage requirement in the out-of-core multifrontal method for sparse factorization. ACM Transactions on Mathematical Software 12, 127–148 (1986)
11. Liu, J.W.H.: The multifrontal method for sparse matrix solution: Theory and Practice. SIAM Review 34, 82–109 (1992)
12. Reid, J.K., Scott, J.A.: An out-of-core sparse Cholesky solver. Technical Report RAL-TR-2006-013, Rutherford Appleton Laboratory (2006) (Revised March 2007)
13. Rothberg, E., Schreiber, R.: Efficient methods for out-of-core sparse cholesky factorization. SIAM Journal on Scientific Computing 21(1), 129–144 (1999)
14. Rotkin, V., Toledo, S.: The design and implementation of a new out-of-Core sparse Cholesky factorization method. ACM Transactions on Mathematical Software 30(1), 19–46 (2004)
15. Schulze, J.: Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods. BIT 41(4), 800–841 (2001)

Parallel Eigensolvers for a Discretized Radiative Transfer Problem*

Paulo B. Vasconcelos[1], Osni Marques[2], and Jose E. Roman[3]

[1] Faculdade de Economia da Universidade do Porto,
Rua Dr. Roberto Frias s/n, 4200-464 Porto, Portugal
pjv@fep.up.pt
[2] Lawrence Berkeley National Laboratory,
1 Cyclotron Road, MS 50F-1650, Berkeley, CA 94720-8139, USA
oamarques@lbl.gov
[3] Instituto ITACA, Universidad Politécnica de Valencia,
Camino de Vera s/n, 46022 Valencia, Spain
jroman@dsic.upv.es

Abstract. In this work we consider the numerical computation of eigenpairs of a matrix derived from integral operators. The matrix is associated to a radiative transfer problem in stellar atmospheres that is formulated by means of a weakly singular Fredholm integral equation defined on a Banach space. We examine direct and iterative parallel strategies for the eigensolution phase, using state-of-the-art numerical methods implemented in publicly available software packages.

Keywords: High performance computing, eigenvalue computations, Fredholm integral equation, weakly singular kernel.

AMS subject classification: 68W10, 65F15, 45B05, 65R20, 65D20, 32A55.

1 Introduction

The solution of many problems in natural sciences and engineering would be unthinkable without the extensive use of approximations and numerical methods. There is a great wealth of such methods, based on either direct or iterative (projection-based) algorithms. Yet, without parallel processing many practical problems would not be solvable or would require an unacceptably large amount of time.

In the present contribution we focus on the numerical computation of eigenpairs of integral operators associated to a radiative transfer problem. This problem is formulated by a weakly singular Fredholm integral equation defined on a

* This work was partially supported by the Portuguese and Spanish governments via an Integrated Action (resp. ref. E-41/07 and ref. HP2006-0004), and partly by the Director, Office of Science, Advanced Scientific Computing Research Program, of the U.S. Department of Energy under contract No. DE-AC02-05CH11231.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 336–348, 2008.

Banach space. The numerical approach used to compute the (cluster of) eigenvalues and associated invariant subspace basis for the integral operator is based on its projection into a finite dimensional subspace. By evaluating the projected problem on a specific basis function, an algebraic eigenvalue problem is obtained; further, the corresponding coefficient matrix is banded.

This work examines the numerical computation of eigenpairs of the matrix of the integral operator using state-of-the-art numerical methods implemented in publicly available software packages. Numerical results using both direct and iterative parallel strategies are presented and discussed.

Section 2 provides a brief description of the problem and the necessary mathematical formalism, from the discretization method used to the projected approximate eigenvalue problem. Section 3 summarizes the computing platforms used for the performance analysis, along with a description of the installed software in each one of the platforms. In Sections 4 and 5, implementation details on the data generation as well as on the numerical methods used are addressed. The former section deals with numerical experiments with the ScaLAPACK library, therefore focusing on direct methods capable of delivering all (or a subset of) eigenpairs of the matrix problem. The latter section is driven by the SLEPc library: a number of state-of-the-art iterative methods are tested, providing insights in the more appropriate methods as well as in parameters specification. Both ScaLAPACK and SLEPc are available in the ACTS Collection of the US Department of Energy (DOE) [1]. The work finishes with conclusions and guidelines for the solution of similar problems in the context of high performance computing.

2 Problem Description

We seek to solve

$$T\varphi = \lambda\varphi \tag{1}$$

where $T : X \rightarrow X$ is an integral operator defined in $X = L^1([0, \tau^*])$, that satisfies

$$(Tx)(\tau) = \frac{\varpi}{2} \int_{\tau^*} E_1(|\tau - \tau^*|)\, x(\tau')\, d\tau', \quad 0 < \tau \leq \tau^* \tag{2}$$

where τ is the optical depth of the stellar atmosphere, τ^* is the optical thickness of the stellar atmosphere, $\varpi \in [0, 1]$ is the albedo, and $E_1 = \frac{\varpi}{2} \int_1^\infty \frac{exp(-\tau\mu)}{\mu} d\mu$ is the first exponential-integral function. We refer the reader to [2,3] for details on the formulation of the problem. In a previous work two of the authors have investigated techniques for the solution of the associated radiative transfer equation [4]. Here we apply direct and iterative parallel strategies to investigate the spectral properties of the problem.

An integral equation of this type can be solved by a discretization mechanism, for instance by projecting it into a finite dimensional subspace. In particular, $T\varphi = \lambda\varphi$ can be approximated by

$$T_m\varphi_m = \lambda_m\varphi_m \tag{3}$$

in the finite dimensional subspace given by an m-dimensional subspace of X spanned by m linearly independent functions $(e_{m,j})_{j=1}^{m}$, X_m. Also, $0 = \tau_{m,0} < \tau_{m,1} < \ldots < \tau_{m,m} = \tau^*$. A nonuniform grid can be used, taking into account the boundary layer at 0. However, a symmetric matrix is obtained if a uniform grid is used instead. Moreover, a symmetrization operation can be applied to the non-symmetric operator matrix.

The projection approximation of T, $T_m : X \to X_m$, is defined by

$$T_m x = \pi_m T x = \sum_{j=1}^{m} \langle x, T^* e^*_{m,j} \rangle e_{m,j}. \tag{4}$$

being π_m the projection in the finite dimensional space. This leads to the eigenvalue problem

$$Ax = \theta x \tag{5}$$

of dimension m where A, large sparse and symmetric, corresponds to the restriction of T_m to X_m. The coefficients of A are given by:

$$\begin{cases} \varpi \left[1 + \frac{1}{d_{i,i-1}} (E_3(d_{i,i-1}) - \frac{1}{2}) \right], & i > j \\ \frac{\varpi}{2d_{i,i-1}} \left[-E_3(d_{i,j}) + E_3(d_{i-1,j}) + E_3(d_{i,j-1}) - E_3(d_{i-1,j-1}) \right], & i \neq j \end{cases} \tag{6}$$

being $d_{i,j} = |\tau_{m,i} - \tau_{m,j}|$, $E_3(\tau) = \int_1^\infty \frac{\exp(-\tau\mu)}{\mu^3} d\mu$, and $E_3(0) = \frac{1}{2}$. For further details, we refer the reader to [2].

For computational purposes, the E_3 function is evaluated according to [5]. Noteworthily, the values of A decay significantly from the diagonal and for practical purposes the matrix can be considered banded.

Our goal is to approximate $T_m \varphi_m = \lambda_m \varphi_m$ by solving an associated symmetric eigenvalue problem $Ax = \theta x$ for large values of m.

3 Test Cases and Computing Platforms

The performance analysis was performed using test problems of different size, which were obtained by varying the resolution of the discretization mesh.

Our tests were performed on three platforms: bassi and jacquard, located at DOE's National Energy Research Center (NERSC), and odin, located at Universidad Politécnica de Valencia. Bassi is an IBM p575 POWER 5 system with 122 8-processor nodes and 32 GB of memory per node. Jacquard is an AMD Opteron cluster with 356 dual-processor nodes, 2.2 GHz processors, 6 GB of memory per node, interconnected with a high-speed InfiniBand network. Odin is a cluster of 55 nodes with dual Pentium Xeon processor at 2 GHz with 1 GB of memory per node and interconnected with a high-speed SCI network with 2-D torus topology. On bassi we used the basic linear algebra subroutines (BLAS) available in the ESSL library, while on jacquard we used the BLAS available in the ACML library. The software installation on odin includes SLEPc 2.3.3, PETSc 2.3.3, PRIMME 1.1, Hypre 2.0, and MUMPS 4.7.3.

For all cases showed in this paper we used $\varpi = 0.75$ (although not shown, tests with larger values of ϖ required similar computing times). We used a relative error $\varepsilon \leq 10^{-12}$ for the stopping criterion of the iterative method in order to obtain solutions as "accurate" as the direct method, and also to perform a more realistic comparison of the algorithms' computational performance.

The test cases that have been used for analyzing the performance of the solvers correspond to matrix dimensions (m) of 4K, 8K, 16K, 32K, 64K, 128K and 216K. The average number of non-zero elements per row is about 75. For example, the matrix of order 4K has 290,668 non-zero elements.

4 Solution Strategy with a Direct Eigensolver

4.1 Numerical Components

In this section we focus on ScaLAPACK's *pdsyevx* [6] for the solution of eigenvalue problem (5). This routine can compute all eigenvalues and (optionally) the corresponding eigenvectors, or only those eigenvalues specified by a range of values or a range of indices. The calculations consist of the following steps: reduction of the input (symmetric matrix) to tridiagonal form, computation of the eigenvalues of the tridiagonal using bisection, computation of the eigenvectors of the tridiagonal using inverse iteration, and (if eigenvectors are required) multiplication of the orthogonal transformation matrix from the reduction to tridiagonal form by the eigenvectors of the tridiagonal. Noticeably, *pdsyevx* may fail to produce orthogonal eigenvectors for tighly clustered eigenvalues. Also, it does not reorthogonalize eigenvectors that are on different processes (the extent of reorthogonalization is determined by the memory available).

ScaLAPACK assumes that the global data has been distributed to the processes with a one or two-dimensional block-cyclic data distribution. In the present work we have generated A using a 1-D block column distribution, i.e. each processor generates a block of columns of A. Since in our case A is banded, the 1-D column distribution is more natural and easier to implement. It would be desirable to take advantage of these properties but ScaLAPACK implements only LU and Cholesky factorizations for band matrices.

4.2 Performance Results

Figures 1 and 2 show the timings for the generation of A, and the eigensolution phase with *pdsyevx*, for matrices of dimension 4K, 8K, 16K and 32K, on up to 128 processors. As expected, the generation of A scales almost ideally when the number of processors is increased. Concerning the eigensolution phase, the timings on bassi refer to the computation of all eigenvalues but no eigenvectors, while on jacquard to the computation of the five largest eigenvalues and corresponding eigenvectors. In both cases, the scaling of *pdsyevx* showed to be satisfactory for the dimensions and number of processors that we have considered. Although the generation of A following a 2-D block cyclic distribution might lead to a

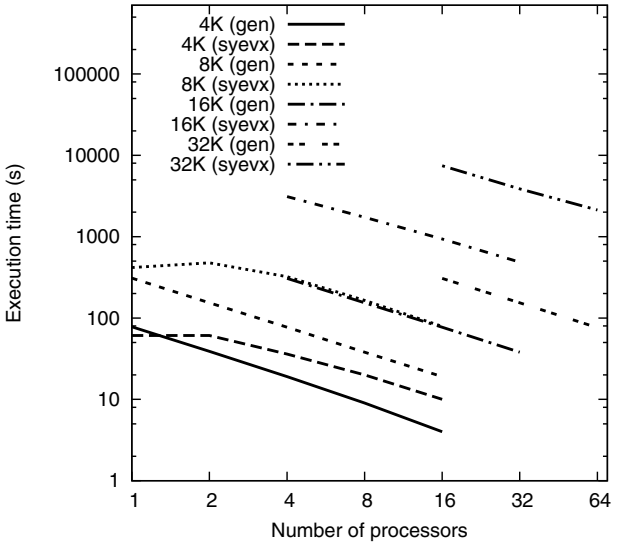


Fig. 1. Execution times for the matrix generation and eigensolution (*pdsyevx*) phases on bassi. The timings are for the computation of all eigenvalues but no eigenvectors.

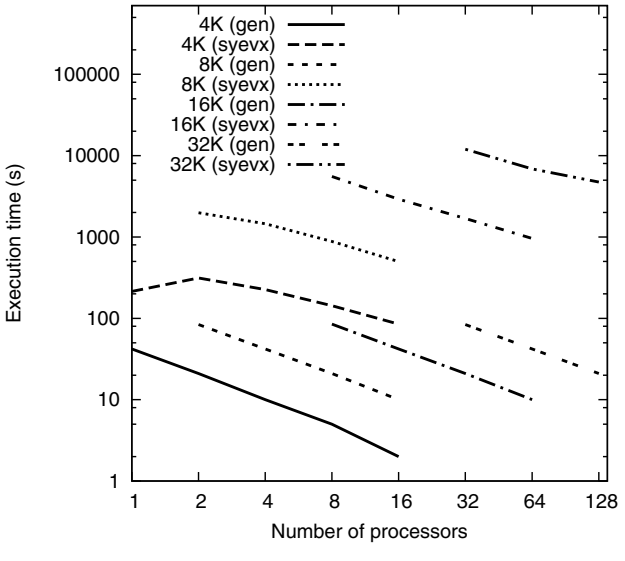


Fig. 2. Execution times for the matrix generation and eigensolution (*pdsyevx*) phases on jacquard. The timings are for the computation of the five largest eigenvalues and corresponding eigenvectors.

better performance, we anticipate that for much larger matrices a direct method would be penalizing. This is also because the matrices become more banded (a property that currently we cannot take advantage of). It is well known that for most cases the reduction to tridiagonal form dominates the costs. However, for some pathological cases, where the eigenvalues are highly clustered, the costs for computing eigenvalues and orthogonalizing them may also be significant [7]. For our problems, the largest eigenvalues become very clustered as the dimension increases. This means that the corresponding eigenvectors would probably have to be orthogonalized more often, therefore increasing the computational costs.

5 Solution Strategy with an Iterative Eigensolver

An iterative eigensolver allows for the solution of larger problems, since one can better exploit the characteristics of the associated matrices (such as sparsity) and no direct transformation of the matrices is needed (such as reduction to tridiagonal form). Also, an iterative eigensolver is usually cheaper than a direct eigensolver when only a subset of eigenvalues and eigenvectors is required. The price to be paid is the convergence rate when there are tightly clustered eigenvalues, as well as the complexity of the different numerical algorithms that have to be used in order to get a scalable solution. Fortunately, there are software tools available that implement those algorithms, and their integration is relatively easy as explained below.

5.1 Numerical Components

PETSc[1], the Portable Extensible Toolkit for Scientific Computation [8], is a parallel framework for the numerical solution of problems arising in applications modeled by partial differential equations. Its design follows an object-oriented approach in order to be able to manage the complexity of numerical methods for very large and sparse problems on parallel computers. It is designed to provide enough flexibility to make software reuse feasible in many different contexts, but also with other goals in mind such as numerical robustness, computational efficiency, portability to different computing platforms, interoperability with other software, etc.

In PETSc all the code is built around a set of objects that encapsulate data structures and solution algorithms. The application programmer works directly with these objects rather than concentrating on the underlying data structures. The data objects include management of index sets, vectors and sparse matrices in different formats, as well as basic support for structured and unstructured meshes. Built on top of this foundation are various classes of solver objects, including linear, nonlinear and time-stepping solvers.

For solving linear systems of equations, PETSc provides a variety of iterative methods such as Conjugate Gradient and GMRES, that can be combined with

[1] PETSc is available at `http://www.mcs.anl.gov/petsc`

different preconditioners such as the incomplete LU factorization. Additionally, direct methods for linear systems are also available via complete factorizations that are handled in a similar way to preconditioners. In both cases, PETSc allows the use of external libraries that are seamlessly integrated in the framework, thus complementing the offered functionality. Examples of such libraries are MUMPS [9] for direct solvers and Hypre for preconditioners, the latter providing different methods such as the algebraic multigrid preconditioner, BoomerAMG [10].

SLEPc[2], the Scalable Library for Eigenvalue Problem Computations [11,12], is a software library for the solution of large, sparse eigenvalue problems on parallel computers. It can be used for the solution of eigenproblems formulated in either standard or generalized form ($Ax = \lambda x$ or $Ax = \lambda Bx$), both Hermitian and non-Hermitian, with either real or complex arithmetic, as well as other related problems such as the singular value decomposition.

SLEPc is built on top of PETSc, and extends it with all the functionality necessary for the solution of eigenvalue problems. It provides uniform and efficient access to a growing number of eigensolvers. Most of these solvers belong to the class of Krylov projection methods, see [13]. In particular, SLEPc implements several variants of the Arnoldi and Lanczos methods, as well as the recently proposed Krylov-Schur method [14] that incorporates a very efficient restarting mechanism. In addition to these solvers, SLEPc seamlessly integrates third-party eigensolver software such as PRIMME [15]. The user is able to easily switch among different eigensolvers by simply specifying the method at run time. Apart from the solver, many other options can be specified such as the number of eigenvalues to compute, the requested tolerance, or the portion of the spectrum of interest.

SLEPc also provides built-in support for different spectral transformations such as the shift-and-invert technique. When such a transformation is applied, the matrix inverses such as $(A-\sigma B)^{-1}$ are not computed explicitly but handled implicitly via a linear system solver provided by PETSc.

For the application described in section 2, the algebraic problem is a standard eigenproblem, that can be symmetric or non-symmetric depending on whether the discretization grid is uniform or not. In the following, we will consider only the symmetric case.

As mentioned in section 2, the sparsity pattern of the matrix is quite special. In fact, it is not really sparse but banded, with a dense band. This will have some implications when treating the matrix from a sparse perspective. In particular, some operations may be quite inefficient with respect to the purely sparse case.

For moderate problem sizes, the Krylov-Schur method does a very good job in computing the largest eigenvalues, i.e. those closest to ϖ, and the corresponding eigenvectors. However, as the problem size grows, it is increasingly difficult for the method to have the solution converged. This difficulty is illustrated in Table 1, with the execution time and number of iterations required to compute 5 eigenpairs of the smallest test cases. In order for the Krylov-Schur method to get the solution in a reasonable number of iterations, it is necessary to increase

[2] SLEPc is available at `http://www.grycap.upv.es/slepc`.

Table 1. Performance of SLEPc and PRIMME on several test cases: m is the matrix size, k is the dimension of the subspace employed by the solver. For the Krylov-Schur and Jacobi-Davidson methods, the required iterations (its) and elapsed time (in seconds) is shown.

	Krylov-Schur			Jacobi-Davidson		
m	k	its	time	k	its	time
4K	48	230	27	48	68	14
8K	96	145	103	48	79	44
16K	192	119	789	48	89	181

the dimension of the subspace used by the solver. The table also shows results for the Jacobi-Davison method implemented in PRIMME, which is able to compute the solution faster and without having to increase the subspace dimension. Unfortunately, for larger test cases both methods fail to compute the solution in a reasonable time.

The problem of slow convergence is due to the fact that eigenvalues get more and more clustered around ϖ as the the order of the matrix grows, and it is well known that convergence of Krylov eigensolvers gets worse when the separation of eigenvalues is poor. Fortunately, the shift-and-invert spectral transformation is a workaround for this problem that fits very well in this application. The idea is to reformulate the eigenproblem as

$$(A - \varpi I)^{-1} x_i = \theta_i x_i. \tag{7}$$

This transformation does not alter the eigenvectors, x_i, and eigenvalues are modified in a simple way, $(\lambda_i - \varpi)^{-1} = \theta_i$, where λ_i are the eigenvalues of the original eigenproblem. If the Krylov-Schur eigensolver is applied to the transformed problem, the eigenvalues closest to ϖ will be retrieved first, as before, the difference being a more favourable separation of eigenvalues.

To illustrate the benefits of shift-and-invert, especially for large matrices, we show some performance data in Table 2. The first eigenvalue is very close to ϖ, and the next ones are very tightly clustered, with a separation of order 10^{-9} in the two largest test cases. With a basis size of 12 vectors, Krylov-Schur requires more than 20,000 restarts to attain the requested tolerance. As mentioned before, increasing the basis size would alleviate this bad convergence, but this is not viable for the largest test cases. In contrast, shift-and-invert performs extremely well, with only three restarts independently of the matrix size, and a total of just 23 linear solves. In the sequel, we will consider only runs of Krylov-Schur with shift-and-invert using a basis size of 12 vectors.

The spectral transformation enhances convergence dramatically, and it is provided by SLEPc in a straightforward manner. The downside is that the code has to deal with an inverted matrix, $(A - \varpi I)^{-1}$. Of course, this matrix is not computed explicitly. Instead, linear solves are performed whenever a matrix-vector product is required by the eigensolver.

Table 2. Performance of SLEPc on several test cases: m is the dimension of the matrix, λ_1 is the largest eigenvalue, sep is the average distance among the 5 largest eigenvalues. For the Krylov-Schur method (K-S) with and without shift-and-invert (S-I), the table shows the required iterations (its) and elapsed time (with MUMPS in the latter case).

			K-S		K-S with S-I	
m	λ_1	sep	its	time	its	time
4K	0.749999813794	$1.12 \cdot 10^{-6}$	21,349	298	3	0.21
64K	0.749999999272	$4.37 \cdot 10^{-9}$	N/A	N/A	3	3.54
128K	0.749999999818	$1.09 \cdot 10^{-9}$	N/A	N/A	3	7.09

Two alternatives exist for solving linear systems: direct and iterative methods. And both alternatives are provided in the context of PETSc, as mentioned before. For the shift-and-invert spectral transformation, it is more natural to use a direct method, providing full accuracy for the computed vector. However, that approach could lead to a non-scalable code or represent an exceedingly high cost. The iterative solver alternative can be a cheap and effective solution, provided that the requested tolerance is sufficient and that the convergence of the inner iteration is guaranteed by a good preconditioner.

Since only a subset of the eigenpairs are computed with SLEPc, the eigensolution stage will be relatively fast and the higher cost will reside in the computation of the matrix elements. Thus, the matrix generation stage has to be effectively parallelized in order to achieve good overall performance.

In PETSc, parallel matrices are distributed by blocks of contiguous rows. That is, a given processor owns a range of matrix rows and stores them in a sparse format. The internal format may vary depending on which solver is going to be used, but this is transparent to the code that sets the matrix elements. For instance, direct solvers such as MUMPS use a particular storage scheme.

In order to exploit the data distribution scheme during the parallel matrix generation, the different processors will compute only the matrix elements that belong to their locally owned rows. With this simple rule, there is no communication involved in the creation of the matrix, and parallel performance should be close to optimal provided that the matrix rows are equally distributed. However, it turns out that the number of non-zero elements is not uniform across the different rows, and this may lead to load imbalance. This imbalance is not severe, though, as will be illustrated in the performance results below. On the other hand, there is no obvious way of predicting how elements decay in a given row of the matrix, so the number of non-zero elements cannot be estimated prior to the actual computation of matrix elements.

In addition to the scheme discussed in the previous paragraph, our actual implementation incorporates the following enhancements:

1. Symmetry is exploited, that is, whenever a matrix element $a_{i,j}$ is computed, it is set also in the position corresponding to the symmetric element, $a_{j,i}$. This reduces the cost of the computation roughly by half. The drawback is

that when the computation is done in parallel, the assembly of the matrix will imply communication among the processors.

2. The computation of each $E_3(d_{i,j})$ can be amortized because it is used in many different matrix elements. It is difficult to determine a priori which are those elements. Nevertheless, there is a locality effect that makes reuse more likely in close matrix elements. In order to exploit this fact, we implemented a simple caching mechanism that stores recently computed values of E_3. With a cache size of only 12 values, the percentage of cache hits in a typical run is 75%, so three quarters of the computation is avoided.

The above optimizations are a detriment to parallel efficiency, but we opted for prioritizing sequential performance and still retain good parallel behaviour.

5.2 Performance Results

For analyzing the parallel performance of the SLEPc-based code, the test problems have been solved on odin, requesting only 5 eigenvalues with a basis size of 12 vectors. In this section, only results are shown corresponding to the largest test case. Despite the large matrix size, the eigensolver does a very good job in getting the solution converged, because of the effectiveness of shift-and-invert in this case.

If a direct linear solver is used for the systems associated to the shift-and-invert transformation, then the computation with one processor is really fast. For instance, with MUMPS the overall eigencomputation takes 7 seconds. However,

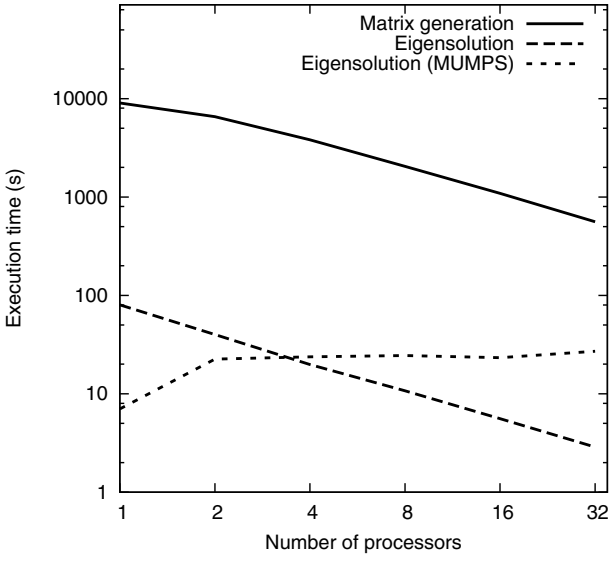


Fig. 3. Execution time in log-log scale for the matrix generation and eigensolution stages with SLEPc corresponding to the 128K test case on odin computer

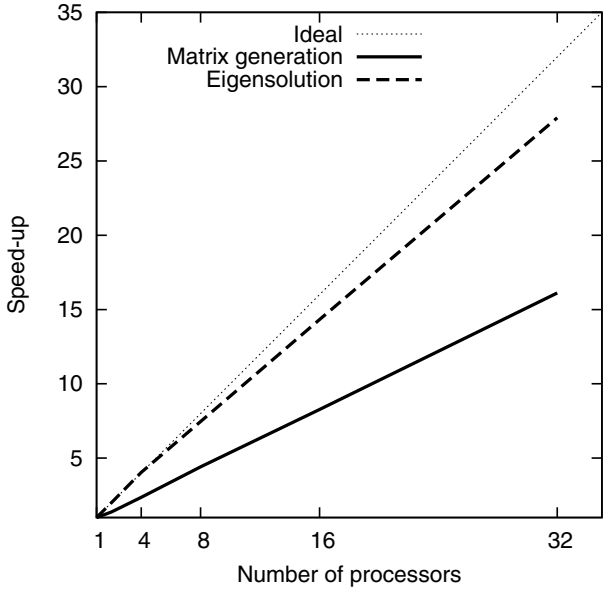


Fig. 4. Speed-up in log-log scale for the matrix generation and eigensolution stages with SLEPc corresponding to the 128K test case on odin computer

the situation is worse in parallel: 22.6 seconds with 2 processors, and this time cannot be reduced with more processors, see the horizontal line in Figure 3. Therefore, we discard the use of direct linear solvers in this setting, because they are not scalable, at least in this type of platforms. It should be stressed that the non-scalability comes from the problem properties, because MUMPS is not optimized for the case of banded, dense matrices.

As discussed before, the alternative is to use an iterative linear solver for the shift-and-invert transformation. In our application, GMRES combined with the algebraic multigrid preconditioner works very well. With one processor the computation is significantly slower (80 seconds), but scalability is remarkably good up to 32 processors and it catches up soon. Figures 3 and 4 show the execution time and the speed-up, respectively. Up to 32 processors, the execution time decreases with constant slope, and speed-up is close to the optimal one. The data for more processors are not shown in these figures, because the number of iterations of the linear system solver changes significantly (e.g. with 40 processors it makes a total of 115 linear iterations, whereas only 70 are necessary up to 32 processors), and thus those times are not comparable. It seems that the preconditioner starts to lose effectiveness when the piece of the matrix assigned to each processor becomes too small.

Regarding the generation of the matrix, Figure 3 shows a time curve that starts with a significantly flatter slope. This is due to the fact that the enhancements discussed before make matrix generation sequentially very efficient. The consequence is a moderate speed-up, as illustrated in Figure 4. Overall, the efficiency of the computation is reasonably good.

Table 3. Scalability of SLEPc. For p processors, a problem of order $m = 4000 \cdot p$ is solved. T_{gen} and T_{solve} are the execution times (in seconds) for matrix generation and eigencomputation, respectively. *its* are the accumulated GMRES iterations.

p	m	T_{gen}	T_{solve}	*its*
1	4K	21	2.70	92
2	8K	40	2.85	92
4	16K	75	2.91	91
8	32K	145	2.91	89
16	64K	294	2.83	82
32	128K	630	2.94	73
54	216K	1012	3.37	86

In order to carry out a fair comparison of the performance for more than 32 processors, we run test cases with increasing problem size, so that the number of matrix rows assigned to each processor remains constant. Table 3 presents the obtained execution times, showing that the time for the eigencomputation phase is more or less constant. The total of accumulated linear solve iterations do not vary too much in this case. We can conclude that the computation of eigenvalues with SLEPc and BoomerAMG scales linearly with the problem size.

6 Conclusions

In this contribution we considered the numerical computation of eigenelements of integral operators associated to a radiative transfer problem, using direct and iterative strategies available in publicly available software packages. The generation of the associated matrices in parallel leads to a significant reduction in computing time. Concerning the eigensolution phase, the algorithm implemented ScaLAPACK's *pdsyevx* showed good scalability for the number of processors used, for computing all eigenvalues only, or a subset of eigenvalues and corresponding eigenvectors. Similarly, SLEPc showed good scalability for the number of processors used for computing a subset of eigenvalues and corresponding eigenvectors. However, a closer look at Figures 1-2 and 3-4 reveals that a direct method becomes more costly as the problem size increases, greatly surpassing the (by itself costly) generation of the matrix. This results from the $O(n^3)$ complexity of the algorithm (in particular the reduction to tridiagonal form) and also because we cannot exploit the band structure of our problems. A strong point in favor of a direct method would be the proper determination of eigenvalue multiplicities. However, we have also been able to deal with such cases with iterative methods. Therefore, for our applications, iterative methods become the method of choice when a subset of eigenvalues is wanted.

References

1. Drummond, L.A., Marques, O.A.: An overview of the advanced computational software (ACTS) collection. ACM Transactions on Mathematical Software 31(3), 282–301 (2005)
2. Ahues, M., d'Almeida, F.D., Largillier, A., Titaud, O., Vasconcelos, P.: An L^1 refined projection approximate solution of the radiation transfer equation in stellar atmospheres. Journal of Computational and Applied Mathematics 140, 13–26 (2002)
3. Rutily, B.: Multiple scattering theoretical and integral equations. In: Cotanda, C., Ahues, M., Largillier, A. (eds.) Integral Methods in Science and Engineering: Analytic and Numerical Techniques, pp. 211–231. Birkhauser, Basel (2004)
4. Marques, O.A., Vasconcelos, P.B.: Evaluation of linear solvers for astrophysics transfer problems. In: Daydé, M., Palma, J.M.L.M., Coutinho, Á.L.G.A., Pacitti, E., Lopes, J.C. (eds.) VECPAR 2006. LNCS, vol. 4395, pp. 466–475. Springer, Heidelberg (2007)
5. Abramowitz, M., Stegun, I.A. (eds.): Handbook of Mathematical Functions with Formulas, Graphs and Mathematical Tables. Applied Mathematics Series, vol. 55. National Bureau of Standards, Washington (1964); Reprinted by Dover, New York
6. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK Users Guide. Society for Industrial and Applied Mathematics, Philadelphia (1997)
7. Dhillon, I.S.: A New $O(N^2)$ Algorithm for the Symmetric Tridiagonal Eigenvalue/Eigenvector Problem PhD. Thesis, University of California, Berkeley (1997)
8. Balay, S., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc users manual. Technical Report ANL-95/11 - Revision 2.3.3, Argonne National Laboratory (2007)
9. Amestoy, P.R., Duff, I.S., L'Excellent, J.Y.: Multifrontal parallel distributed symmetric and unsymmetric solvers. Computer Methods in Applied Mechanics and Engineering 184(2–4), 501–520 (2000)
10. Henson, V.E., Yang, U.M.: BoomerAMG: A parallel algebraic multigrid solver and preconditioner. Applied Numerical Mathematics: Transactions of IMACS 41(1), 155–177 (2002)
11. Hernandez, V., Roman, J.E., Vidal, V.: SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. ACM Transactions on Mathematical Software 31(3), 351–362 (2005)
12. Hernandez, V., Roman, J.E., Tomas, A., Vidal, V.: SLEPc users manual. Technical Report DSIC-II/24/02 - Revision 2.3.3, D. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia (2007)
13. Bai, Z., Demmel, J., Dongarra, J., Ruhe, A., van der Vorst, H. (eds.): Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide. Society for Industrial and Applied Mathematics, Philadelphia (2000)
14. Stewart, G.W.: A Krylov–Schur algorithm for large eigenproblems. SIAM Journal on Matrix Analysis and Applications 23(3), 601–614 (2001)
15. Stathopoulos, A.: Nearly optimal preconditioned methods for Hermitian eigenproblems under limited memory. Part I: Seeking one eigenvalue. SIAM Journal on Scientific Computing 29(2), 481–514 (2007)

High Performance Computing and the Progress of Weather and Climate Forecasting

Philippe Bougeault

European Centre for Medium-Range Weather Forecasts, UK

Abstract. Over the last few years, global weather forecasts from a number of operational numerical weather prediction centres have continued to progress steadily in skill and reach an impressive level of quality. The drivers of progress are increased resolution, better numerical algorithms, massive amounts of new observations from satellites, and very advanced data assimilation schemes to ingest observations in models. All of these aspects rely heavily on the ever increasing computing resources that have become available. The first part of the talk will describe some aspects of these improvements and will give considerations on how much each type of improvement has actually contributed to the overall improvement in forecast skill.

While the most advanced weather forecasting models have reached global horizontal resolutions of about 20km, climate models are rather ranging from 300km to 100km. Recent research points to the need to increase the resolution of climate models to benefit from the same large improvements as the weather forecast models have undergone. In fact the concept of seamless weather and climate forecasts is rapidly developing. The second part of the talk will focus on this concept.

The third part of the talk will highlight current difficulties in developing highly scalable systems, based on the ECMWF example. The ECMWF high performance computer system consists of 2 IBM POWER5+ Cluster 1600 supercomputer systems. Each cluster is made up of 155 compute nodes, plus a further 10 nodes which are used for I/O and networking. Each node consists of 8 Dual-core Power5+ processors, ie 16 physical cores, which have a clock frequency of 1.9 GHz., a peak performance of 7.6 Gflops/core and a sustained performance of 1 Gflop/core. Simultaneous Multi-Threading (SMT) means that each physical core can run 2 threads concurrently. So with SMT switched on, each node can be considered to have 32 logical cores. Users specify jobs by the number of MPI processes and the number of OpenMP threads. I will show examples of work to improve the scalability of various parts of our forecasting system on this machine.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, p. 349, 2008.

Simulation of Seismic Wave Propagation in an Asteroid Based upon an Unstructured MPI Spectral-Element Method: Blocking and Non-blocking Communication Strategies

Roland Martin[1], Dimitri Komatitsch[1,2], Céline Blitz[1], and Nicolas Le Goff[1]

[1] Université de Pau et des Pays de l'Adour,
CNRS & INRIA Sud-Ouest Magique-3D,
Laboratoire de Modélisation et d'Imagerie en Géosciences UMR 5212,
Avenue de l'Université, 64013 Pau Cedex, France
{roland.martin,dimitri.komatitsch,celine.blitz,
nicolas.legoff}@univ-pau.fr
http://www.univ-pau.fr
[2] Institut universitaire de France, 103 boulevard Saint-Michel, 75005 Paris, France
{dimitri.komatitsch}@univ-pau.fr
http://www.cpu.fr/Iuf/

Abstract. In order to better understand the internal structure of asteroids orbiting in the Solar system and then the response of such objects to impacts, seismic wave propagation in asteroid 433-Eros is performed numerically based on a spectral-element method at frequencies lying between 2 Hz and 22 Hz. In the year 2000, the NEAR Shoemaker mission to Eros has provided images of the asteroid surface, which contains numerous fractures that likely extend to its interior. Our goal is to be able to propagate seismic waves resulting from an impact in such models. For that purpose we create and mesh both homogeneous and fractured models with a highly-dispersive regolith layer at the surface using the CUBIT mesh generator developed at Sandia National Laboratories (USA). The unstructured meshes are partitioned using the METIS software package in order to minimize edge cuts and therefore optimize load balancing in our parallel blocking or non-blocking MPI implementations. We show the results of several simulations and illustrate the fact that they exhibit good scaling.

Keywords: Non-blocking MPI, load balancing, scaling, mesh partitioning, seismic wave propagation, asteroids.

1 Introduction

In the context of seismic exploration, it is of crucial importance to develop efficient numerical tools to model seismic wave propagation in complex structures with great accuracy at scales of at least tens of kilometers. For this purpose, the seismic wave equation can be solved numerically in heterogeneous media using

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 350–363, 2008.

different methods, for instance the finite-difference technique or the boundary integral technique. In the last decade, the spectral-element method (SEM), which is more accurate and flexible, has been used extensively in the context of regional or global seismology. The SEM was introduced twenty years ago in computational fluid mechanics by [1]. It is a high-order variational method that retains the ability of finite elements to handle complicated geometries while keeping the exponential convergence rate of spectral methods. Complex topographies, dipping or curved interfaces, interface and surface waves, and distorted meshes can be easily taken into account. Indeed, the SEM provides a more natural context to describe the free surface of the model thanks to the weak formulation of the equations that is used, for which the free surface condition is a natural condition. The formulation of the SEM based on the displacement vector and on Gauss-Lobatto-Legendre numerical integration which is implemented in our software package called SPECFEM has the property to retain a diagonal mass matrix and is therefore easier to implement than classical low-order finite-element methods. Applications of the SEM to two-dimensional (2D) (e.g., [3,4]) and three-dimensional (3D) elastodynamics (e.g., [5,6]) have shown that high accuracy (i.e., small numerical dispersion) is obtained. The time discretization is an explicit conditionally-stable second-order centered finite-difference scheme. Because of the diagonal mass matrix and of the standard explicit time scheme, no inversion of a linear system is needed and therefore the method can be efficiently implemented in parallel and large 3D models can be handled (e.g., [5,6,7]). In this article, we use the SEM to simulate seismic wave propagation resulting from an impact at the surface of a 2D cut plane in an asteroid.

Asteroids are metallic or rocky bodies orbiting in the Solar System. Very little is known about their internal structure and therefore several models of their interior have been proposed, for instance monoliths (objects of low porosity, good transmitters of elastic stress) or rubble pile (shattered bodies whose pieces are grouped into a loose and porous packing [8]). Depending on its structure, the response of an asteroid to an impact (for instance when it is hit by a meteor or by an artificial impactor sent on it at high velocity) can be very different [9]: a rubble pile would be harder to disrupt than a monolith [8]. Thus, to develop mitigation techniques (to prevent a potential collision of a hazardous object with the Earth) it is important to be able to perform simulations of body disruption on reliable models of an asteroid interior [10]. The geophysical knowledge of asteroid interiors can be improved by space missions sent to comets or asteroids. In the year 2000, the NEAR Shoemaker mission to 433-Eros has provided images of the asteroid surface, which displayed numerous fractures and evidence of a coherent but fractured interior [11]. The study of the crater distribution at the surface of Eros has highlighted a deficit in the distribution of small crater sizes. To explain this observation, [12] have proposed impact-induced vibrations as a possible source of downslope movements on crater walls. The related mobilized material could fill the smallest craters, leading to their erasure. In this study we therefore simulate wave propagation in different models of the interior of asteroid 433-Eros.

2 Spatial and Temporal Discretization of the Governing Equations

We consider a linear isotropic elastic rheology for the heterogeneous solid, and therefore the seismic wave equation can be written in the strong form as:

$$\begin{aligned} \rho\ddot{\mathbf{u}} &= \boldsymbol{\nabla}\cdot\boldsymbol{\sigma}+\mathbf{f}\ , \\ \boldsymbol{\sigma} &= \mathbf{C}:\boldsymbol{\varepsilon}=\lambda\mathrm{tr}(\boldsymbol{\varepsilon})\mathbf{I}+2\mu\boldsymbol{\varepsilon}\ , \\ \boldsymbol{\varepsilon} &= \tfrac{1}{2}[\boldsymbol{\nabla}\mathbf{u}+(\boldsymbol{\nabla}\mathbf{u})^T]\ , \end{aligned} \tag{1}$$

where $\mathbf{u}$ denotes the displacement vector, $\boldsymbol{\sigma}$ the symmetric, second-order stress tensor, $\boldsymbol{\varepsilon}$ the symmetric, second-order strain tensor, $\mathbf{C}$ the fourth-order stiffness tensor, λ and μ the two Lamé parameters, ρ the density, and $\mathbf{f}$ an external force. The trace of the strain tensor is denoted by $\mathrm{tr}(\boldsymbol{\varepsilon})$, $\mathbf{I}$ denotes the identity tensor, the tensor product is denoted by a colon, and a superscript T denotes the transpose. A dot over a symbol indicates time differentiation. The physical domain is denoted by Ω, and its outer boundary by Γ. The material parameters of the solid, $\mathbf{C}$ (or equivalently λ and μ) and ρ, can be spatially heterogeneous. We can then rewrite the system (1) in a variational weak form by dotting it with an arbitrary test function $\mathbf{w}$ and integrating by parts over the whole domain as:

$$\int_\Omega \rho\mathbf{w}\cdot\ddot{\mathbf{u}}\,\mathrm{d}\Omega+\int_\Omega \boldsymbol{\nabla}\mathbf{w}:\mathbf{C}:\boldsymbol{\nabla}\mathbf{u}\,\mathrm{d}\Omega=\int_\Omega \mathbf{w}\cdot f\,\mathrm{d}\Omega+\int_\Gamma(\boldsymbol{\sigma}\cdot\hat{\mathbf{n}})\cdot\mathbf{w}\,\mathrm{d}\Gamma\ . \tag{2}$$

The free surface (i.e. traction free) boundary condition on Γ is easily implemented in the weak formulation since the integral of traction along the boundary simply vanishes (e.g., [5]) when we set $\boldsymbol{\tau}=\boldsymbol{\sigma}\cdot\hat{\mathbf{n}}=0$ at the free surface.

This formulation is solved on a mesh of quadrangular elements in 2D, which honors both the free surface of the asteroid and its main internal discontinuities (for instance its fractures). The unknown wave field is expressed in terms of high-degree Lagrange polynomials on Gauss-Lobatto-Legendre interpolation points, which results in an exactly diagonal mass matrix that leads to a simple time integration scheme (e.g. Komatitsch et al., 2005). Let $\mathbf{w}_N$, $\mathbf{u}_N$ denote the piecewise-polynomial approximations of the test functions and the displacement, respectively. Making use of (2), the discrete variational problem to be solved can thus be expressed as: for all time t, find $\mathbf{u}_N$ such that for all $\mathbf{w}_N$ we have

$$\int_\Omega \rho\mathbf{w}_N\cdot\ddot{\mathbf{u}}_N\,\mathrm{d}\Omega+\int_\Omega \boldsymbol{\nabla}\mathbf{w}_N:\mathbf{C}:\boldsymbol{\nabla}\mathbf{u}_N\,\mathrm{d}\Omega=\int_\Omega \mathbf{w}_N\cdot f\,\mathrm{d}\Omega\ . \tag{3}$$

We can rewrite this system (3) in matrix form as:

$$M\ddot{d}+Kd=F\ , \tag{4}$$

where M is the diagonal mass matrix, F is the source term, and K is the stiffness matrix. For detailed expression of these matrices, the reader is referred for instance to [6].

Time discretization of the second-order ordinary differential equation (4) is achieved using the explicit Newmark central finite-difference scheme [5,6], which is second order accurate and conditionally stable :

$$M\ddot{\mathbf{d}}_{n+1} + K\mathbf{d}_{n+1} = F_{n+1} \ , \tag{5}$$

where

$$\mathbf{d}_{n+1} = \mathbf{d}_n + \Delta t\dot{\mathbf{d}}_n + \frac{\Delta t^2}{2}\ddot{\mathbf{d}}_n \ , \tag{6}$$

and

$$\dot{\mathbf{d}}_{n+1} = \dot{\mathbf{d}}_n + \frac{\Delta t}{2}[\ddot{\mathbf{d}}_n + \ddot{\mathbf{d}}_{n+1}] \ . \tag{7}$$

At the initial time $t = 0$, null initial conditions are assumed i.e., $\mathbf{d} = \mathbf{0}$ and $\dot{\mathbf{d}} = \mathbf{0}$. The time step Δt and the distribution of mesh sizes $h(x, y)$ are chosen such that the Courant-Friedrichs-Lewy (CFL) stability condition and a numerical dispersion condition are satisfied. The CFL condition is :

$$max\left(\frac{c_p(x,y)}{h(x,y)}\right)\Delta t \leq \alpha \ , \tag{8}$$

where $c_p(x, y)$ is the pressure velocity distribution in the model under study, and the numerical dispersion condition is

$$min\left(\frac{c_s(x,y)}{h(x,y)}\right)/(2.5f_0) \geq n_\lambda \ , \tag{9}$$

where f_0 is the dominant frequency of the seismic source, $c_s(x, y)$ the shear velocity distribution in the model and n_λ the minimum number of grid points per seismic wavelength. In practice α is taken lower than 0.5, and n_λ about 5 [13].

3 Model of the Asteroid

The size of Eros is approximately 34 km (length) by 17 km (height) and we have designed its model according to the dataset provided by the NEAR Shoemaker spacecraft. These data display a regolith blanket at the surface (formed by crushed rocks during impacts) of thickness ranging from tens to hundreds of meters [11]. We therefore added a regolith blanket around the asteroid, with a thickness ranging from 50 to 150 m. The images of the surface of Eros also display numerous craters and fractures. These fractures are thought to be formed by impacts and the regolith depression found around them suggests regolith infiltration [11]. This regolith depression can be up to 300 m wide, for instance near the Rahe Dorsum ridge. This long fracture, striking on Eros images, probably crosses the asteroid interior and is therefore included in our 2D model. Except Rahe Dorsum, all fractures have been designed under the main craters in our 2D model and filled with the same material as the regolith. Because of technical constraints, we designed simple fracture shapes, and to avoid low angles in the

Table 1. The quality of the quadrangle elements can be defined by their angles θ (rad) or equivalently by their skewness, which is defined as $|(2\theta - \pi)/\pi|$. Ideal angles are right angles (90 degrees) with skewness of 0, and poor angles are lower than typically 30 degrees with skewness beyond approximately 0.65. Here the number of poor elements is small and their influence on the global calculations is therefore expected to remain reasonable.

Model	Homogeneous (3744 elements)	Fractured (57275 elements)
Average angle	84.14	81.98
Standard deviation angle	5.57	7.44
Min Angle	59.83	29.76
Max angle	89.94	89.97
Average skew	0.0606	0.0861
Standard deviation skew	0.0575	0.0925
Min skew	0	0
Max skew	0.356	0.647

mesh elements, we tried to draw the fracture geometry with angles close to 90 degrees. We extended the fracture network to a depth of one crater radius or more according to [16].

We define two models of the interior of Eros: one that is homogeneous (it includes the topography of Eros and an elastic material characterized by a pressure wave velocity $c_p = 3000$ m.s^{-1}, a shear wave velocity $c_s = 1700$ m.s^{-1}, and a density = 2700 kg.m^{-3}), and another one that in addition comprises a regolith layer as well as fracture networks. The interior of the fractures and the regolith layer have the same material properties: c_p= 900 m.s^{-1}, c_s= 500 m.s^{-1}, and a density of 2000 kg.m^{-3}. Seismic attenuation (i.e., loss of energy by viscoelasticity) can be ignored in this problem because seismic studies performed on the Moon have shown that it is negligible.

The two models are meshed with CUBIT quadrangular and hexahedral mesh generator developed at Sandia National Laboratories (*http://cubit.sandia.gov*, USA). Table 1 shows that the quality of the meshes obtained for the two models displayed in Figure 1 is good: the angles of the elements are acceptable (comprised between approximately 30 and 90 degrees) with a related skewness lying between 0 and 0.65 (0 corresponds to the best case of a perfectly cubic element and 1 corresponds to the worst case of a flat element). For speed-up scaling purposes we perform different calculations by increasing the number of mesh points and elements while we increase the dominant frequency of the seismic source according to the dispersion relation. Subsequently the number of partitions increases accordingly, each mesh partition being mapped to one processor core. Dominant frequencies are chosen from 2 Hz to 22 Hz and element sizes take values of 18 m (in fractures and regolith layers) to 522 m (in the bedrock) at 2 Hz, and 1.68 m (fractures and regolith) to 47.5 m (bedrock) at 22 Hz. Chosing higher frequencies is not relevant because the available data resolution (300 m) as well as the interior structure are not known accurately enough from a physical point of view.

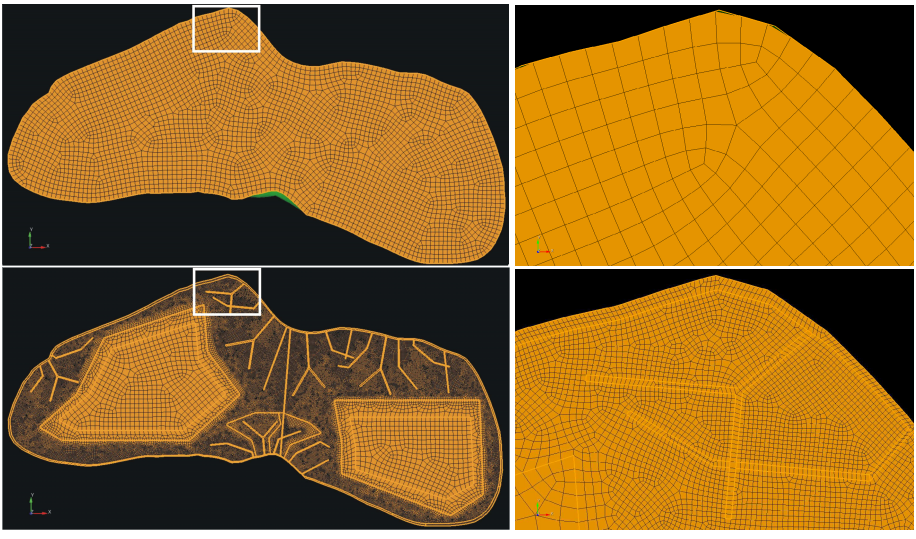


Fig. 1. Meshes created using the CUBIT mesh generator for an homogeneous model of asteroid Eros (top, 3744 elements) and a more complex model with a regolith layer and a network of fractures (bottom, 52275 elements) for simulations performed at a central frequency of 2 Hz. Close-ups on the white frame are also shown.

4 Mesh Partitioning and Non-blocking MPI Implementation

A 2D mesh designed for a very high-resolution simulation is too large to fit on a single computer. We therefore implement the calculation in parallel based upon MPI. We first partition the mesh using the METIS graph partitioning library [14], which focuses on balancing the size of the different domains and minimizing the edge cut. Figure 2 shows how the mesh for the homogeneous model is partitioned by METIS into 8 or into 80 domains. Balancing the size of the domains ensures that no processor core will be idle for a significant amount of time while others are still running at each iteration of the time loop, while a small edge cut reduces the number and the size of the communications. Contributions between neighboring elements that are located on different processor cores are added using non-blocking MPI sends and receives and following a similar implementation for low-order finite element method described in [15]. The communication scheme used is the following: contributions (i.e., mechanical internal forces) from the outer elements of a given mesh slice (i.e., elements that have an edge in common with an element located on a different processor core) are computed first and sent to the neighbors of that mesh slice using a non-blocking MPI send. Similarly, each processor core issues non-blocking MPI receives. This allows for classical overlapping of the communications by the calculations, i.e. each process then has time to compute mechanical forces in its inner elements

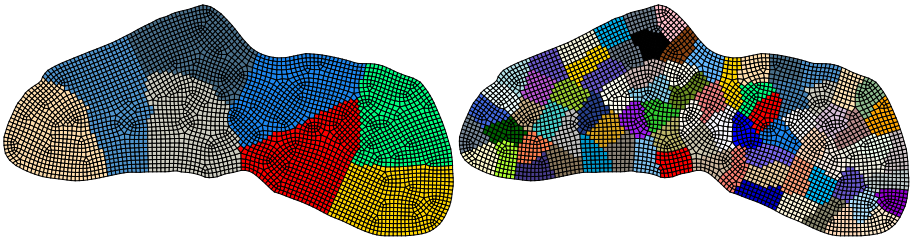


Fig. 2. Partitioning of the mesh for the homogeneous model (Figure 1) obtained with METIS in the case of 8 (left) and 80 (right) domains. We observe that the number of elements along the interface of the partitions is small compared to the number of elements inside each partition in the case of 8 domains, and therefore overlapping of communications with calculations is expected to work fine. But in the case of 80 domains, this number becomes comparable or even higher than the number of inner elements, in which case overlapping is expected to fail, and poor performance should result.

while the communications travel across the network, and if the number of outer elements is small compared to the number of inner elements in all the mesh slices we can be confident that the messages will be arrived when the internal calculations are finished. Once the contributions from neighbors have been received, they are added to the corresponding elements locally. We ran the code on a Dell PowerEdge 1950 cluster with Intel EM64T Xeon 5345 (Clovertown) processors (2333 MHz clock frequency, 9.330 Gigaflops peak performance) and Myrinet network, located at the California Institute of Technology (USA). Each processor is dual core, so we prefer to speak here in terms of processes or processor cores with one process per processor core.

5 Numerical Simulations and Scaling of the Code

In all the simulations we use a polynomial of degree $N = 4$ to integrate variables at the $(N + 1)^2 = 25$ Gauss-Lobatto-Legendre points along each direction of any spectral element. A series of simulations to measure speedup and scaling are performed at a dominant frequency of 2 Hz for the homogeneous model first and then for the fractured model. A third series of tests are computed to study the weak scaling of the code. By weak scaling we mean how the time to solve a problem with increasing size can be held constant by enlarging the number of processes used and by maintaining a fixed system size per process: when one doubles the number of processes one also doubles the system size.

Moderate mesh sizes are considered for the homogeneous model, and moderate to large mesh sizes for the fractured model (Table 1). The seismic source is an impact represented by a force normal to the surface and located at point (x = -9023 m, y = 6131 m) in the mesh. The time variation of the source is the second derivative of a Gaussian. The time step used is $\Delta t = 1$ ms and the signal is propagated for 150000 time steps (i. e. 150 s). In the case of the homogeneous

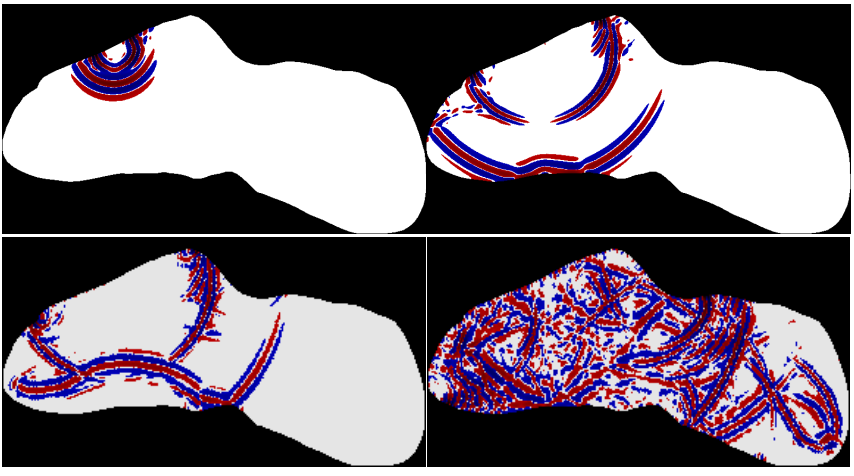


Fig. 3. Snapshots of the propagation of simulated seismic waves in the asteroid for a total duration of 65 seconds in the case of the mesh for a homogeneous model displayed in Figure 1. Snapshots are shown at 10 s, 25 s (top) and 45 s, 65 s (bottom). We represent the vertical component of the displacement vector in red (positive) or blue (negative) at each grid point when it has an amplitude higher than a threshold of 1% of the maximum, and the normalized value is raised to the power 0.30 in order to enhance small amplitudes that would otherwise not be clearly visible.

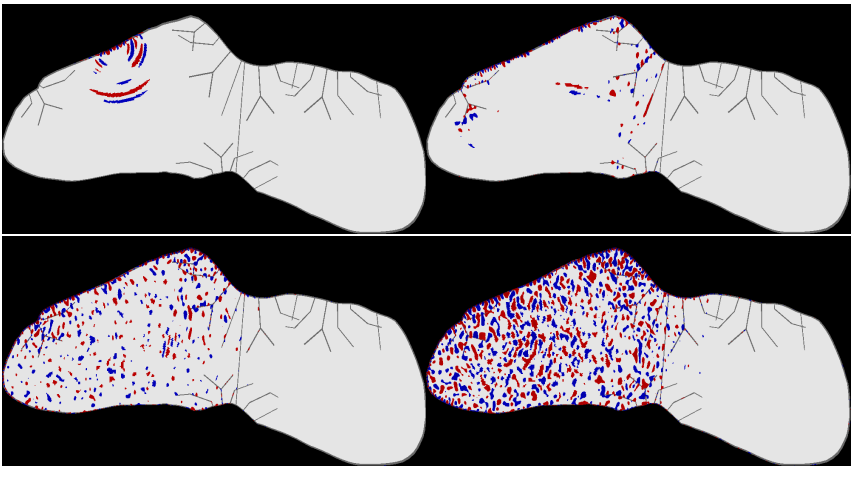


Fig. 4. Snapshots of the propagation of simulated seismic waves in the asteroid for a total duration of 65 seconds in the case of the mesh for the more complex model with a regolith layer and networks of fractures displayed in Figure 1. Snapshots are shown at 10 s, 25 s (top) and 45 s, 65 s (bottom). We represent the vertical component of the displacement vector in red (positive) or blue (negative) at each grid point when it has an amplitude higher than a threshold of 1% of the maximum, and the normalized value is raised to the power 0.30 in order to enhance small amplitudes that would otherwise not be clearly visible.

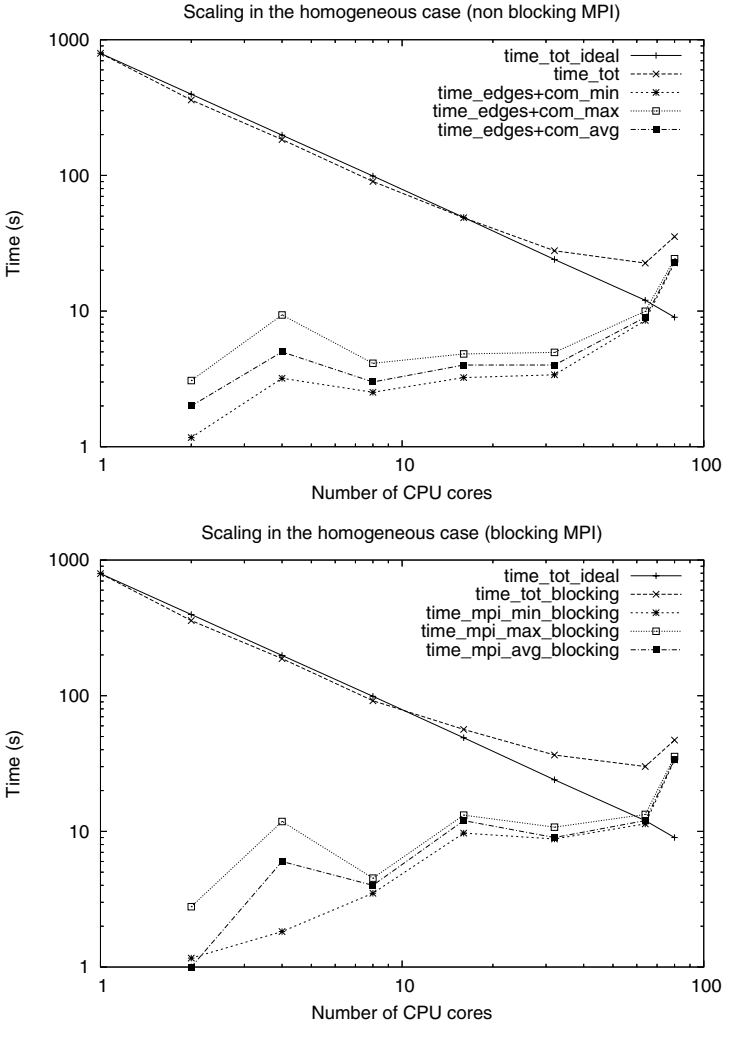


Fig. 5. Scaling of the code in the case of the mesh for the homogeneous model represented in Figure 1 for blocking (top) and non blocking communications (bottom). For both strategies the measured total time spent in the calculations and communications (long dashed line with single crosses) is compared to the theoretical straight line obtained assuming perfect scaling (solid line) for different numbers of processes (one per mesh partition): 1, 2, 4, 8, 32, 64 and 80. The two curves are in very good agreement as long as the number of outer elements is small compared to the number of inner elements in all the mesh slices, in which case overlapping of communications with calculations using non-blocking MPI works very well. Here for the relatively small mesh used this is true when we use a number of processes lower than 32. When we use a larger number of processes scaling quickly becomes very poor because communications are no longer overlapped. One can also observe that the total time spent in communications and calculations is similar between blocking and non-blocking communication strategies.

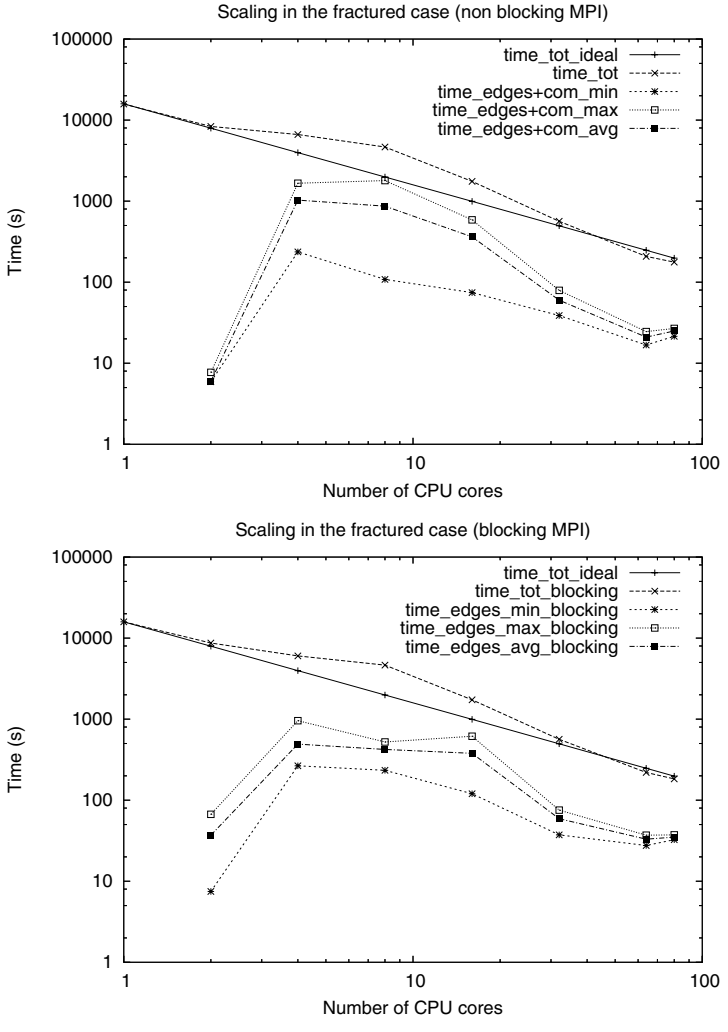


Fig. 6. Scaling of the code in the case of the mesh for the fractured model represented in Figure 1 for blocking (top) and non blocking communication (bottom) strategies. For both strategies the measured total time spent in the calculations and communications (long dashed line with single crosses) is compared to the theoretical straight line obtained assuming perfect scaling (solid line) for different numbers of processor cores (one per mesh partition): 1, 2, 4, 8, 32, 64, 80. The two curves are in very good agreement as long as the number of outer elements is small compared to the number of inner elements in all the mesh slices, in which case overlapping of communications with calculations using non-blocking MPI works very well. Here for the relatively small mesh used this is always the case. We do not observe the same poor scaling as in the homogeneous case but it should start to appear if we used more than 80 processes. Total time spent in communications and calculations is not really different between blocking and non-blocking communication procedures for this specific application, as observed in the homogeneous case.

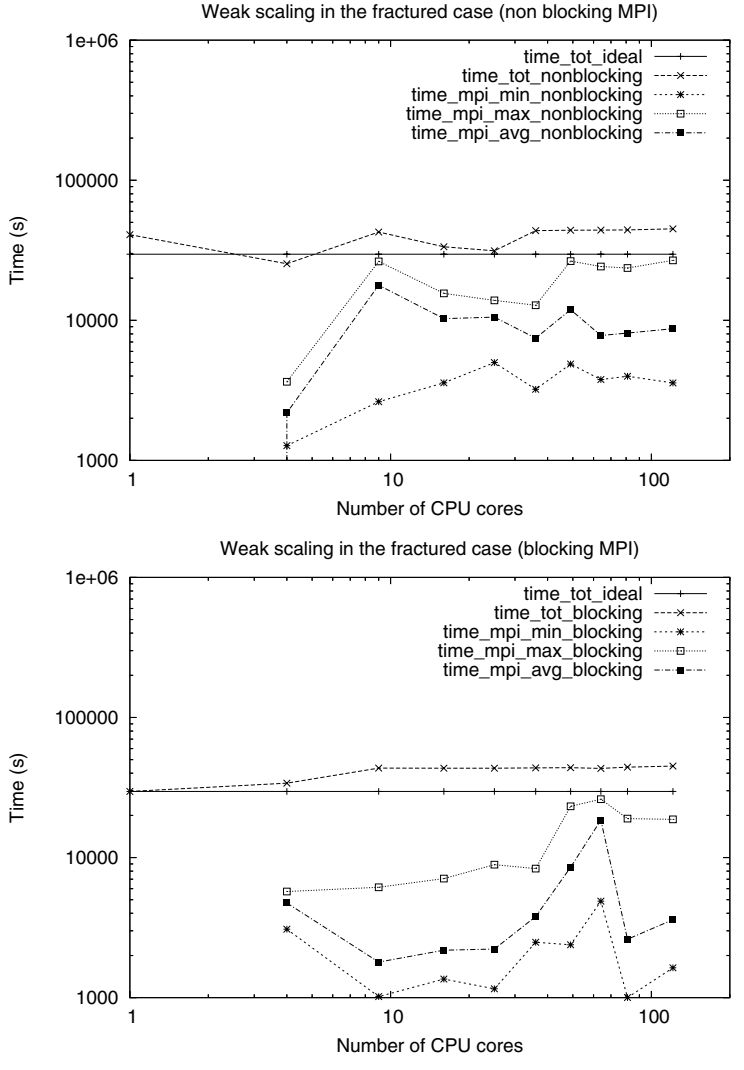


Fig. 7. Weak scaling of the code for the fractured model represented in Figure 1 in the case of non blocking (top) and blocking (bottom) communications. The measured total time spent in the calculations and communications (long dashed line with single crosses) is compared to the theoretical straight line obtained assuming perfect scaling (solid line) for different numbers of processor cores (one per mesh partition) and different dominant frequencies : 1 (2 Hz and 57275 elements), 4, 9, 16, 25, 36, 49, 64, 81, 121 (22 Hz and around 110 million points) processes. As we increase the number of processes and the corresponding seismic frequency, the number of elements increases until reaching more than 110 million points for 121 processes. It is interesting to note that the total computational time spent per process remains constant for more than 10 processes in the non-blocking case. Similar results are obtained for blocking communications.

model, Figure 3 shows pressure, shear and surface waves propagating inside Eros and along its surface for 65 seconds. One can see in particular the pressure wave front that is reflected on the lower free surface and the shear wave front that closely follows the pressure wave front, as well as many reflections and conversions of waves generated along the whole surface of the object. On the contrary, in the fractured model (figure 4), waves travel mostly inside the dispersive surface layer of regolith as well as the fractures, which act as wave guides. The waves then reflect off all the boundaries and are trapped for a while in the left part of the model, where the source is located, and then progressively move to the right part, the large transversal fracture acting as a geological barrier.

Figure 5 shows the scaling of the code for 1 to 80 processes in the case of the mesh for the homogeneous model with 3744 elements with sizes from 147 m to 470 m. As expected, scaling is very good (close to the straight line obtained for hypothetical perfect scaling) as long as the number of outer elements is small compared to the number of inner elements in all the mesh slices, in which case overlapping of communications with calculations using non-blocking MPI works very well. Here for the relatively small mesh used this is true when we use a number of processes lower than 32. When we use a number of processes that is larger and for which this assumption ceases to be true, scaling quickly becomes very poor because communications are no longer overlapped. Indeed, for more than 32 processes the total measured time goes through a minimum and then starts to increase. The three other curves (medium and tiny dashed lines and filled square lines), which represent the lowest, average and highest values of the sum of the time spent in calculations and communications in the outer elements of all the partitions, increase when we use more than 32 processes, which explains the poor scaling observed in this case. It is observed that total time spent in communications and calculations is not really different between blocking and non-blocking communication procedures for this specific application.

In the fractured case, scaling is performed for 57275 spectral elements (919709 grid points) at 2 Hz by increasing the number of processes from 1 to 121. Figure 6 shows that the scaling is much better than for the homogeneous case. The poor scaling that appears beyond 32 processes in the homogeneous case will appear for a much higher number of processes in this heterogeneous case. This is not surprising because the number of elements is higher inside each partition.

Now, instead of increasing the number of processes for a given fixed mesh, a weak scaling study is performed for the complex and highly unstructured meshes generated for the fractured model. The dominant frequency of the source is increased from 2 Hz to 22 Hz while we refine the mesh in the same ratio and increase the number of processor cores consequently from 1 to 121. By construction of the cracks in the asteroid and using a mesh decimation technique in each element, skewnesses and angles are reasonably preserved when the number of elements increases. Time steps and mesh sizes are respectively decreased from 0.45 ms (2 Hz) to 0.04 ms (22 Hz). At 2 Hz, mesh sizes take values varying from 18 m (regolith) to 522 m (bedrock), while at 22 Hz they vary from 1.68 m to 47 m. In Figure 7 we observe that the total computational time per

process is nearly constant as the number of processes increases beyond more than 10 processes approximately. This is observed for both blocking or non-blocking communication strategies, which give similar results. The tests correspond to 57275 elements (nearly 1 million points at 2 Hz) for 1 processor core and to nearly 6 million elements (nearly 111 million points at 22 Hz) for 121 processor cores. The number of degrees of freedom (the two components of the displacement vector) is exactly twice the number of points, i.e., close to 222 millions. Let us finally mention that a mixed MPI/OpenMP model could also be used but Komatitsch et al. [7] have shown that it does not bring any significant gain in performance for this particular application.

6 Conclusions

We have simulated wave propagation in a homogeneous or a fractured model of an asteroid represented by a non-structured mesh. A mesh with good skewness has been developed with CUBIT. For both blocking and non-blocking communication strategies using METIS, similar scalings are obtained and mesh configurations of 110 million points can be computed using 121 processor cores and dominant seismic frequencies of up to 22 Hz. In future work it would be interesting to extend the simulations to 3D at high resolution and to apply L2 cache misses reduction techniques [17].

Acknowledgments. The authors would like to thank Philippe Lognonné for fruitful discussion about asteroids, Emanuele Casarotti and Steven J. Owen for fruitful discussion about meshing with CUBIT, Jean Roman and Jean-Paul Ampuero for fruitful discussion about overlapping communications with non-blocking MPI. Calculations were performed on the Division of Geological & Planetary Sciences Dell cluster at the California Institute of Technology (USA). This material is based in part upon research supported by European FP6 Marie Curie International Reintegration Grant MIRG-CT-2005-017461 and by the French ANR under grant NUMASIS ANR-05-CIGC-002.

References

1. Patera, A.T.: A spectral element method for fluid dynamics: laminar flow in a channel expansion. Journal of Computational Physics 54, 468–488 (1984)
2. Cohen, G., Joly, P., Tordjman, N.: Construction and analysis of higher-order finite elements with mass lumping for the wave equation. In: Kleinman, R. (ed.) Proceedings of the second international conference on mathematical and numerical aspects of wave propagation, pp. 152–160. SIAM, Philadelphia (1993)
3. Priolo, E., Carcione, J.M., Seriani, G.: Numerical simulation of interface waves by high-order spectral modeling techniques. Journal of Acoustical Society of America 95(2), 681–693 (1994)
4. Komatitsch, D., Martin, R., Tromp, J., Taylor, M.A., Wingate, B.A.: Wave propagation in 2-D elastic media using a spectral element method with triangles and quadrangles. Journal of Computational Acoustics 9(2), 703–718 (2001)

5. Komatitsch, D., Vilotte, J.P.: The spectral-element method: an efficient tool to simulate the seismic response of 2D and 3D geological structures. Bull. Seis. Soc. Am. 88(2), 368–392 (1998)
6. Komatitsch, D., Tromp, J.: Introduction to the spectral-element method for 3-D seismic wave propagation. Geophy. J. Int. 139(3), 806–822 (1999)
7. Komatitsch, D., Tsuboi, S., Ji, C., Tromp, J.: A 14.6 billion degrees of freedom, 5 teraflops, 2.5 terabyte earthquake simulation on the Earth Simulator. In: Proceedings of the ACM/IEEE Supercomputing SC 2003 conference, CD-ROM (2003), www.sc-conference.org/sc2003
8. Asphaug, E.: Interior structures for asteroids and cometary nuclei. In: Belton, M.J.S., Morgan, T.H., Samarasinha, N., Yeomans, D.K. (eds.) Mitigation of Hazardous Comets and Asteroids, pp. 66–103. Cambridge University Press, Cambridge (2004)
9. Holsapple, K.A.: About deflecting asteroids and comets. In: Belton, M.J.S., Morgan, T.H., Samarasinha, N., Yeomans, D.K. (eds.) Mitigation of Hazardous Comets and Asteroids, pp. 113–140. Cambridge University Press, Cambridge (2004)
10. Michel, P., Benz, W., Richardson, D.C.: Disruption of fragmented parent bodies as the origin of asteroids families. Nature 421, 608–611 (2003)
11. Robinson, M.S., Thomas, P.C., Veverka, J., Murchie, S.L., Wilcox, B.B.: The geology of 433 Eros. Meteoritics and planetary sciences 37, 1651–1684 (2002)
12. Richardson, J.E., Melosh, H.J., Greenberg, R.J., O'Brien, D.P.: The global effects of impact-induced seismic activity on fractured asteroid surface morphology. Icarus 179, 325–349 (2005)
13. De Basabe, J.D., Sen, M.K.: Grid dispersion and stability criteria of some common finite-element methods for acoustic and elastic wave equations. Geophysics 72(6), 81–95 (2007)
14. Karypis, G., Kumar, V.: Multilevel k-way Partitioning Scheme for Irregular Graphs. J. Parallel Distrib. Comput. 48(1), 96–129 (1998)
15. Danielson, K.T., Namburu, R.R.: Nonlinear dynamic finite element analysis on parallel computers using Fortran 1990 and MPI. Advances in Engineering Software 29(3-6), 179–186 (1998)
16. Ahrens, T.J., Xia, K., Coker, D.: Depth of cracking beneath impact craters: new constraint for impact velocity. In: Furnish, M.D., Thadhani, N.N., Horie, Y. (eds.) Shock-Compression of Condensed Matter, pp. 1393–1396. American Institute of Physics, New York (2002)
17. Komatitsch, D., Labarta, J., Michea, D.: A simulation of seismic wave propagation at high resolution in the inner core of the Earth on 2166 processors of MareNostrum. In: Proceedings of the VecPar 2008 8th International Meeting on High Performance Computing for Computational Science, Toulouse, France, June 24-27 (2008) (in press)

A Simulation of Seismic Wave Propagation at High Resolution in the Inner Core of the Earth on 2166 Processors of MareNostrum

Dimitri Komatitsch[1,2], Jesús Labarta[3], and David Michéa[1]

[1] Université de Pau et des Pays de l'Adour,
CNRS UMR 5212 & INRIA Sud-Ouest Magique-3D,
Avenue de l'Université, 64013 Pau Cedex, France
{dimitri.komatitsch,david.michea}@univ-pau.fr
http://www.univ-pau.fr
[2] Institut universitaire de France, 103 boulevard Saint-Michel, 75005 Paris, France
http://www.cpu.fr/Iuf
[3] Barcelona Supercomputing Center, Technical University of Catalonia,
c/ Jordi Girona, 31 - 08034 Barcelona, Spain
jesus@cepba.upc.es
http://www.bsc.es

Abstract. We use 2166 processors of the MareNostrum (IBM PowerPC 970) supercomputer to model seismic wave propagation in the inner core of the Earth following an earthquake. Simulations are performed based upon the spectral-element method, a high-degree finite-element technique with an exactly diagonal mass matrix. We use a mesh with 21 billion grid points (and therefore approximately 21 billion degrees of freedom because a scalar unknown is used in most of the mesh). A total of 2.5 terabytes of memory is needed. Our implementation is purely based upon MPI. We optimize it using the ParaVer analysis tool in order to significantly improve load balancing and therefore overall performance. Cache misses are reduced based upon renumbering of the mesh points.

Keywords: load balancing, cache misses, mesh partitioning, seismic wave propagation, global Earth.

1 Introduction

Modeling of seismic wave propagation resulting from large earthquakes in the three-dimensional (3D) Earth is of considerable interest in seismology because analyzing seismic wave propagation in the Earth is one of the few ways of studying the structure of the Earth's interior, based upon seismic tomography. Seismic waves resulting from earthquakes can be classified in two main categories: body waves, which propagate inside the medium and are of two types: compressional (pressure) waves, called P waves, and shear waves, called S waves; and surface waves, which travel along the surface of the medium and have an exponentially decreasing amplitude with depth. The analysis of the 3D geophysical structure of

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 364–377, 2008.

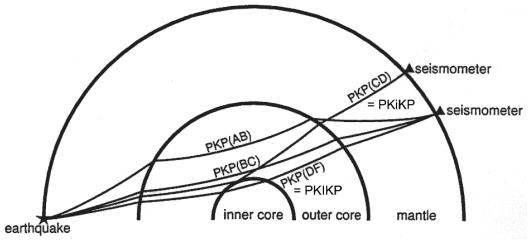


Fig. 1. Sketch of how PKP seismic phases propagate in the Earth after an earthquake, i.e. illustration of their ray paths. They therefore carry information about the anisotropic structure of the inner core.

the Earth therefore requires the ability to calculate accurate numerical seismograms (time series representing the three component of displacement at points located on the surface of the Earth). In particular, pressure waves can be used to study the solid inner core of the Earth and its anisotropy (i.e. varying seismic wave speed in different spatial directions). Figure 1 shows a sketch of PKP seismic phases, which are pressure waves that travel inside the core of the Earth. They travel through the Earth's mantle, then through its fluid outer core and solid inner core, then again in the mantle, and then reach the surface, where they are recorded.

The field of numerical modeling of seismic wave propagation in 3D geological media has significantly evolved in the last decade due to the introduction of the spectral-element method (SEM), which is a high-degree version of the finite-element method that is very accurate for linear hyperbolic problems such as wave propagation, having very little intrinsic numerical dispersion. It combines the flexibility of the finite-element method with the accuracy of the pseudospectral method. In addition, the mass matrix is exactly diagonal by construction, which makes it much easier to implement on parallel machines because no linear system needs to be inverted. The 3D SEM was first used in seismology for local and regional simulations (e.g., [1,2,3]) and then adapted to wave propagation at the scale of the full Earth (e.g., [4,5]). Until recently, at the scale of the global Earth available computer resources intrinsically limited such large calculations. For instance, on a PC cluster with 150 processors, Komatitsch and Tromp [4] reached a maximum seismic frequency of 0.0555 Hz, and on 1944 processors of the Japanese Earth Simulator Komatitsch et al. [6] reached a maximum seismic frequency of 0.2 Hz. Such frequencies are not high enough to capture important differential effects on seismic wave propagation related to anisotropy in the inner core of the Earth. Here we implement the SEM on MareNostrum, the world's number 13 supercomputer as of the November 2007 Top500 list of supercomputers, which is located in Barcelona, Catalonia, Spain. We show that on 2166 of its IBM PowerPC 970 processors we can simulate seismic waveforms accurately up to a maximum frequency of 0.5 Hz based upon message passing with MPI.

2 Spatial and Temporal Discretization of the Governing Equations

We consider a linear anisotropic elastic rheology for the heterogeneous solid Earth, and therefore the seismic wave equation can be written in the strong (i.e., differential) form as:

$$\begin{aligned} \rho\ddot{\mathbf{u}} &= \boldsymbol{\nabla}\cdot\boldsymbol{\sigma}+\mathbf{f}\ , \\ \boldsymbol{\sigma} &= \mathbf{C}:\boldsymbol{\varepsilon}\ , \\ \boldsymbol{\varepsilon} &= \tfrac{1}{2}[\boldsymbol{\nabla}\mathbf{u}+(\boldsymbol{\nabla}\mathbf{u})^T]\ , \end{aligned} \tag{1}$$

where $\mathbf{u}$ denotes the displacement vector, $\boldsymbol{\sigma}$ the symmetric, second-order stress tensor, $\boldsymbol{\varepsilon}$ the symmetric, second-order strain tensor, $\mathbf{C}$ the fourth-order stiffness tensor, ρ the density, and $\mathbf{f}$ an external force. The tensor product is denoted by a colon, a superscript T denotes the transpose, and a dot over a symbol indicates time differentiation. The physical domain of the model is denoted by Ω and its outer boundary by Γ. The material parameters of the solid, $\mathbf{C}$ and ρ, can be spatially heterogeneous. We can then rewrite the system (1) in a weak (i.e., variational) form by dotting it with an arbitrary test function $\mathbf{w}$ and integrating by parts over the whole domain:

$$\int_\Omega \rho\mathbf{w}\cdot\ddot{\mathbf{u}}\,\mathrm{d}\Omega+\int_\Omega \boldsymbol{\nabla}\mathbf{w}:\mathbf{C}:\boldsymbol{\nabla}\mathbf{u}\,\mathrm{d}\Omega=\int_\Omega \mathbf{w}\cdot f\,\mathrm{d}\Omega+\int_\Gamma(\boldsymbol{\sigma}\cdot\hat{\mathbf{n}})\cdot\mathbf{w}\,\mathrm{d}\Gamma\,. \tag{2}$$

The free surface (i.e., traction free) boundary condition is easily implemented in the weak formulation since the integral of traction $\boldsymbol{\tau}=\boldsymbol{\sigma}\cdot\hat{\mathbf{n}}$ along the boundary simply vanishes when we set $\boldsymbol{\tau}=\mathbf{0}$ at the free surface of the Earth.

This formulation is solved on a mesh of hexahedral elements in 3D, which honors both the free surface of the model and its main internal discontinuities (i.e., its geological layers). The unknown wave field is expressed in terms of high-degree Lagrange polynomials of degree N on Gauss-Lobatto-Legendre interpolation (GLL) points, which results in a diagonal mass matrix that leads to a simple time integration scheme (e.g., [1,3]). Because that matrix is diagonal, no linear system needs to be inverted and the method lends itself well to calculations on large parallel machines with distributed memory. Let $\mathbf{w}_N$, $\mathbf{u}_N$ denote the piecewise-polynomial approximations of the test functions and the displacement respectively. Making use of (2), the discrete variational problem to be solved can thus be expressed as: for all time t, find $\mathbf{u}_N$ such that for all $\mathbf{w}_N$ we have:

$$\int_\Omega \rho\mathbf{w}_N\cdot\ddot{\mathbf{u}}_N\,\mathrm{d}\Omega+\int_\Omega \boldsymbol{\nabla}\mathbf{w}_N:\mathbf{C}:\boldsymbol{\nabla}\mathbf{u}_N\,\mathrm{d}\Omega=\int_\Omega \mathbf{w}_N\cdot f\,\mathrm{d}\Omega\,. \tag{3}$$

We can rewrite this system (3) in matrix form as:

$$M\ddot{d}+Kd=F\ , \tag{4}$$

where M is the diagonal mass matrix, F is the source term, and K is the stiffness matrix. For detailed expression of these matrices, the reader is referred for instance to [3].

Time discretization of the second-order ordinary differential equation (4) with a time step Δt is achieved using the explicit Newmark central finite-difference scheme [7] which is second-order accurate and conditionally stable :

$$M\ddot{\mathbf{d}}_{n+1} + K\mathbf{d}_{n+1} = F_{n+1} \ , \tag{5}$$

where

$$\mathbf{d}_{n+1} = \mathbf{d}_n + \Delta t\dot{\mathbf{d}}_n + \frac{\Delta t^2}{2}\ddot{\mathbf{d}}_n \ \text{ and } \ \dot{\mathbf{d}}_{n+1} = \dot{\mathbf{d}}_n + \frac{\Delta t}{2}[\ddot{\mathbf{d}}_n + \ddot{\mathbf{d}}_{n+1}] \ . \tag{6}$$

At the initial time $t = 0$, null initial conditions are assumed i.e., $\mathbf{d} = \mathbf{0}$ and $\dot{\mathbf{d}} = \mathbf{0}$. The stability condition is $\Delta t(c_p/\Delta x)_{\max} \leq c$, where Δx is the distance between two adjacent grid points, c_p is the speed of the pressure waves in the geological medium, and c is a constant that is of the order of 0.5 [8].

3 Implementation of the Spectral-Element Method on MareNostrum

3.1 Acoustic/Elastic Formulation

We are interested in differential effects on very high frequency (0.5 Hertz) seismic phases when they propagate inside the solid inner core of the Earth, therefore to significantly reduce the computational cost we suppress the crust of the Earth and replace it with an extended upper mantle, and convert the whole mantle from elastic to acoustic, thus reducing the problem in that part of the model from a vectorial unknown to a scalar unknown, i.e. reducing memory usage and CPU cost by a factor of roughly three in 3D. In the acoustic mantle and crust we solve the acoustic wave equation in terms of a fluid potential [4]. We keep a (much more expensive to solve) elastic anisotropic medium in the inner core only. In that small part of the mesh we also model seismic attenuation (i.e., loss of energy by viscoelasticity), which has a significant impact on the cost of that small part of the simulation because memory requirements increase by a factor of roughly 2 and CPU time by a factor of roughly 1.5 [4].

3.2 Mesh Generation

Figure 2 shows a global view at the surface of the Earth of the spectral-element mesh we designed, which is accurate up to a frequency of 0.5 Hertz for pressure waves and which fits on 2166 processor cores (6 blocks of 19 × 19 slices). The sphere is meshed using hexahedra only, based upon an analytical mapping from the six sides of a unit cube to a six-block decomposition of the surface of the sphere called the 'cubed sphere' [9,4,5]. Each of the six sides of the 'cubed sphere' mesh is divided into 19 × 19 slices, shown with different colors, for a total of 2166 slices. We allocate one slice and therefore one MPI process per processor core (which means that in the remainder of the article when we say 'one processor' for simplicity we actually mean 'one processor core').

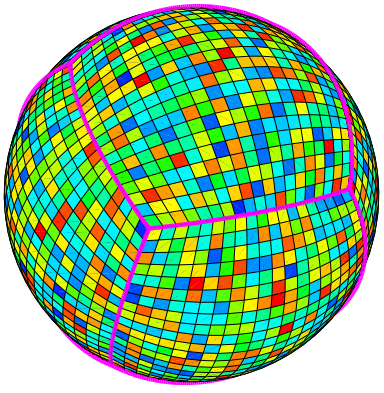


Fig. 2. The SEM uses a mesh of hexahedral finite elements on which the wave field is interpolated by high-degree Lagrange polynomials on Gauss-Lobatto-Legendre integration points. The figure shows a global view of the mesh at the surface, illustrating that each of the six sides of the so-called 'cubed sphere' mesh is divided into 19 × 19 slices, shown here with different colors, for a total of 2166 slices (i.e., one slice per processor core).

The total number of spectral elements in this mesh is 323 millions, which corresponds to a total of approximately 21 billion global grid points (the 'equivalent' of a 2770 × 2770 × 2770 grid), because each spectral element contains $(N+1)^3 = 5 \times 5 \times 5 = 125$ grid points since we use polynomial basis functions of degree $N = 4$, but with points on its faces shared by neighboring elements. This in turn also corresponds to approximately 21 billion degrees of freedom because a scalar unknown is used in most of the mesh (everywhere except in the inner core of the Earth, as mentioned above). Such simulations use a total of approximately 2.5 terabytes of memory.

Our SEM software package is called SPECFEM3D. Version 1.0 was released in 1999 and the current stable version is 3.6. In order to be able to run our large-scale calculations on MareNostrum, we had to fix some significant load balancing issues in version 3.6 and therefore produce a new version called 4.0. Below we discuss the main improvements made.

3.3 Type of Operations Performed at Each Time Step

At each iteration of the serial time loop, which are all identical, four main types of operations are performed:

- update of global arrays, for instance: for each i from 1 to Npoint,
displacement[i] += Δt velocity[i] + Δt^2 acceleration[i] / 2,
where displacement, velocity and acceleration are three global arrays, and Npoint is the number of grid points

- relatively large and purely local calculations of the product of predefined derivation matrices with a local copy of the displacement vector along cut planes

in the three directions (i, j and k) of a 3D spectral element, which contains $(N+1)^3 = 125$ points; therefore, each index i, j or k varies between 1 and N+1

- element-by-element products and sums of arrays of dimension (N+1,N+1, N+1,Nspec), where Nspec is the number of spectral elements, which involve a quadruple loop on the dimensions of the arrays

- sum of the contributions (which are elemental force vectors from a physical point of view) computed locally in arrays of dimension (N+1,N+1,N+1,Nspec) into global arrays of dimension Npoint using indirect addressing. This sum is called the 'assembly' process in finite elements.

3.4 Exploiting Locality

Increasing and exploiting locality of memory references is an important optimization technique. Locality must be optimized in loops that tend to reference arrays or other data structures by indices. The principle of locality deals with the process of accessing a single resource multiple times; in particular regarding temporal locality (a resource referenced at one point in time will be referenced again sometime in the near future) and spatial locality (the likelihood of referencing a resource is higher if a resource in the same neighborhood has been referenced). Memory should therefore be accessed sequentially as often as possible.

In the spectral-element method these are important issues because, as can be seen in Figure 3 drawn in 2D, each spectral element contains $(N+1)^2 = 25$ GLL points, and points lying on edges or corners (as well as on faces in 3D) are shared between elements. The contributions to the global system of degrees of freedom,

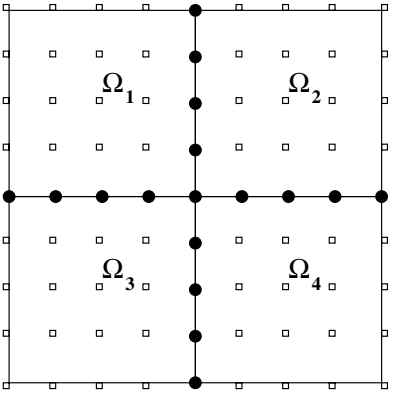


Fig. 3. Illustration of the local and global meshes for a four-element 2D spectral-element discretization with a polynomial degree of $N = 4$. Each element contains $(N+1)^2 = 25$ Gauss-Lobatto-Legendre points. Points lying on edges or corners (as well as on faces in 3D) are shared between elements. The contributions to the global system of degrees of freedom, computed separately on each element, have to be summed at the common points represented by black dots. Corners can be shared by any number of elements depending on the topology of the mesh, which is in most cases non structured.

computed separately on each element, therefore have to be summed at the common points, and the corners can be shared by any number of elements depending on the topology of the mesh, which is in most cases (included ours) non structured. For instance in 3D for a regular hexahedral mesh there are $(N+1)^3 = 125$ GLL integration points in each element, of which 27 belong only to this element (21.6%), 54 belong to two elements (43.2%), 36 belong to four elements (28.8%) and eight belong to 8 elements (6.4%). Hence, 78.4% of the points belong to at least two elements and it is therefore interesting to reuse these points by keeping them in the cache. During mesh creation we must therefore optimize the global numbering which maps point (i,j,k) of local element num_element to a unique global point number (after removal of the duplicated common points shared by more than one element) called global_addressing(num_element, i, j, k).

To do this, the mesh is created element by element, then the common points are identified, and a global addressing is created. This array must be reordered once and for all in the mesher to optimize the future memory access order of the points and elements in the solver, in order to maximize spatial and temporal locality and to access memory sequentially as often as possible. Simple renumbering based on looping on the elements in the mesher in the order in which they will be accessed in the solver and masking points already found works fine for this problem. A more sophisticated approach is to also change the order of the elements (in addition to the numbering of the points) based on the classical reverse Cuthill-McKee algorithm [10] to reduce the average memory strides to access the points shared by several elements, but improvements are very small in the case of the spectral-element method because one spectral element fits entirely in the Level 1 (L1) cache, several tens of spectral elements fit in the L2 cache and a large number of operations are performed on each spectral element because it contains 125 points. Detailed tests not shown here have shown us that it is not necessary in practice to use this algorithm for our problem.

The elementary contributions (internal mechanical forces) from each mesh element are computed based upon products of cut planes in the 3D memory block representing the element with a matrix called the 'derivation matrix' in order to compute the derivative of a given vector or scalar field. We therefore thought about using Basic Linear Algebra Subroutines (BLAS3) 'sgemm' calls in version 4.0 of the code instead of the Fortran loops used in version 3.6. However, this turned out to be inefficient for several reasons. First, these matrix-matrix products are performed on 2D cut planes that are extracted from different (orthogonal) directions of a given 3D memory block. Therefore, in order to use BLAS3 we need to perform some memory copies from the 3D blocks to 2D matrices for some (but not all) of the BLAS3 calls, in order for the input matrix to be correctly structured, which induces significant overhead. Second, these matrices are very small in each element (5×5 or 25×5) and therefore the matrix operations are too small to be efficiently replaced by BLAS3 calls because the overhead is large. Even if we compute all the elements (whose number is large in each mesh slice, typically more than 100,000) simultaneously with one BLAS3 call, we are still dealing with the multiplication of a 5×5 matrix with a

$5 \times 500{,}000$ matrix, a situation for which BLAS3 usually does not perform very well. Third, because we use static loop sizes in the solver (the only drawback being that we need to recompile the solver every time we change the size of the mesh), at compile time the compiler knows the size of all the loops and can therefore very efficiently optimize them (using unrolling for instance). Because the inner loops are very small (of size N+1 = 5), it is very difficult to do better than loop unrolling performed automatically by the compiler. Therefore it is better in terms of performance to let the compiler optimize the static loops rather than to switch to BLAS3.

One way of improving performance is to manually use the Altivec/VMX vector unit of the PowerPC, which can handle four single-precision floating-point operations in a vector and is therefore well suited for our small matrix products since we can load a vector unit with 4 floats, perform several 'multiply-and-add' (vec_MADD) operations to compute the matrix-matrix product, and store the results in four consecutive elements of the result matrix. Since our matrices are of size 5×5 and not 4×4, we use vector instructions for 4 out of each set of 5 values and compute the last one serially in regular Fortran. To improve performance and get correct results we align our 3D blocks of $5 \times 5 \times 5 = 125$ floats on 128 in memory using padding with three dummy values, which induces a negligible waste of memory of 128 / 125 = 2.4% (non aligned accesses lead to incorrect results in Altivec). We typically gain between 15% and 20% in CPU time with respect to version 4.0 without Altivec.

3.5 MPI Implementation and Load Balancing

Our SEM solver is based upon a pure MPI implementation. A few years ago on the Japanese Earth Simulator we implemented a mixed MPI – OpenMP solution, using MPI between nodes (i.e., between blocks of 8 processors with shared memory) and OpenMP inside each node. However, in practice, tests on a small number of processors gave a CPU time that was almost identical to a pure MPI run, and therefore we decided to permanently switch to pure MPI [6]. We do not claim that this conclusion is general; it might well be specific to our SEM algorithm, in particular we did not try the mixed OpenMP – MPI solution on a large number of nodes or on MareNostrum. Other groups have successfully implemented algorithms based upon mixed OpenMP – MPI models on large parallel machines.

I/O is not an issue in our simulations because we only output a small number of time series (called 'seismograms') to record seismic motion (the three components of the displacement vector) at a small number of points at the surface of the mesh. This means that the amount of data saved by our SEM is small.

Figure 4 shows that a regular mesh has the undesirable property that the size of the elements decreases dramatically with depth. To maintain a relatively constant number of grid points per wavelength, element size should increase with depth. In version 3.6 of SPECFEM3D, this was accomplished in three stages based on a simple 'doubling brick'. Each block has four sides that need to match up exactly with four other blocks to complete the cube. Schematically, these four

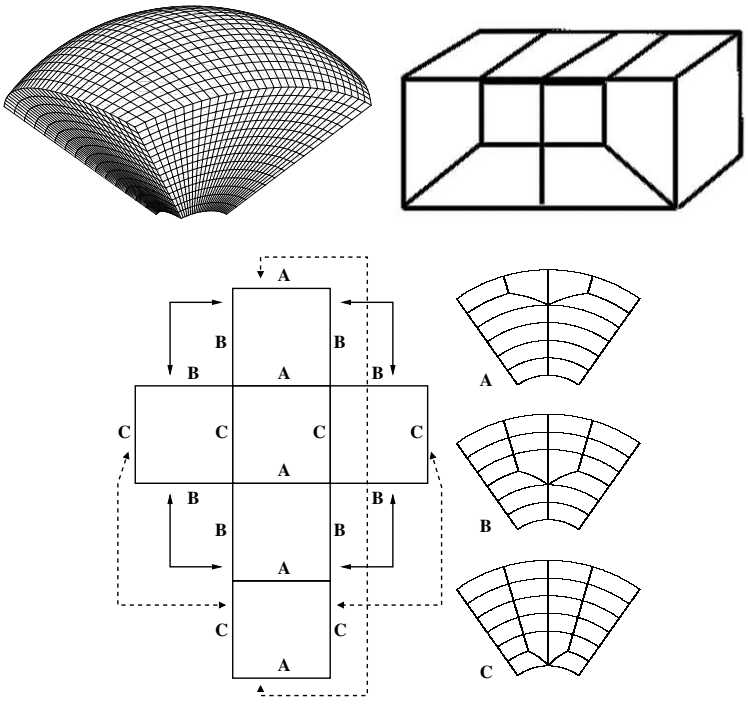


Fig. 4. A regular mesh (top left) has the undesirable property that the size of the elements decreases dramatically with depth. To maintain a relatively constant number of grid points per wave length, element size should increase with depth. In version 3.6 of SPECFEM3D, this is accomplished in three stages based on a simple 'doubling brick' (top right). Each block has four sides that need to match up exactly with four other blocks to complete the cube (bottom), as indicated by the arrows. Schematically, these four sides have one of three designs: A, B, or C, as illustrated on the right. When the six blocks are assembled to make the entire globe, they match perfectly. Unfortunately, the fact that the three types of blocks have a significantly different number of mesh elements induces significant load imbalance.

sides have one of three designs: A, B, or C. When the six blocks are assembled to make the entire globe, they match perfectly. However, because with that simple 'doubling brick' doubling the mesh in two directions in the same layer is topologically impossible, the three mesh types A, B and C contain a significantly (15% to 20%) different number of mesh elements, which in turn results in load imbalance in the same amount because in the SEM one performs the same number of elementary calculations in each element. In addition, an analysis of version 3.6 performed with the ParaVer analysis tool (Figure 5) also showed significant load imbalance in terms of the number of Level 2 (L2) data cache misses in each mesh slice (this will be further discussed below). ParaVer (see e.g. [11] and www.cepba.upc.es/paraver) is a tool developed at the Barcelona Supercomputing Center that is designed to analyze the number of data cache misses and of

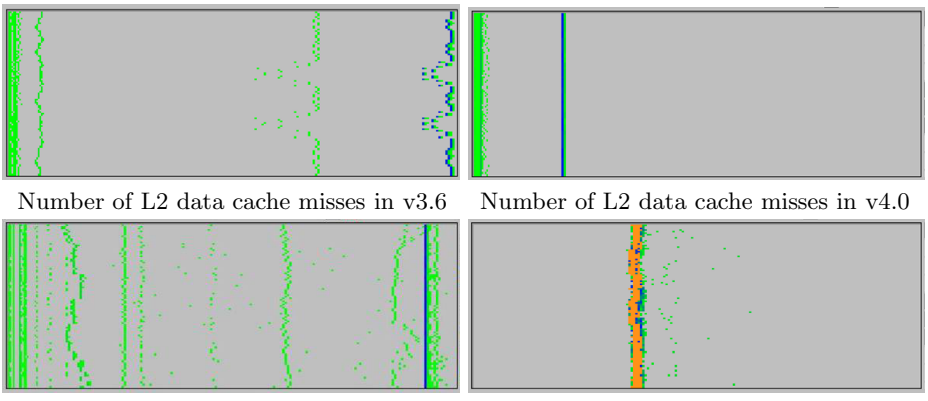


Fig. 5. ParaVer analysis of the code on 96 processors, from processor 1 at the top of each picture to processor 96 at the bottom. Top left: In version 3.6 of SPECFEM3D, L2 cache misses were very poorly balanced between mesh slices (irregular blue and green curve), thus inducing severe load imbalance. In version 4.0 (top right), L2 cache misses (represented on the same horizontal scale) have been drastically reduced and very well balanced (straight blue line). The number of instructions executed is also very well balanced (bottom left, straight blue line). As a result, useful duration of the calculations (bottom right, orange points) is well balanced too.

instructions of MPI processes as well as the useful duration of calculations performed, among many other things. Let us mention that the imbalance observed in terms of cache misses was also observed between slices belonging to the same chunk type (A, B or C) and was therefore mostly due to the numbering of the mesh points (i.e., the order in which they were accessed) and not only to the different mesh structure between different chunks.

In order to address the first issue of geometrical mesh imbalance, in version 4.0 the mesh doubling of Figure 4 is now accomplished in only one level instead of two based on a more efficient geometrical 'doubling brick' which is assembled in a symmetric block of four 'doubling bricks' based on mirror symmetry of the basic brick (Figure 6). This makes it possible to carry out the doubling in both directions in the same layer. As a result, while in the old mesh of version 3.6 there are three types of mesh chunks (labeled A, B and C in Figure 4), in version 4.0 of the code this poor property of the mesh is suppressed and all the mesh chunks have the same shape and exact same number of elements, thus resulting in perfect geometrical mesh balancing. Because the number of operations performed in each element is the same, load balancing is therefore very significantly improved.

In order to have more uniform mesh sampling in the inner core of the Earth, in version 4.0 we also slightly inflate the central cube in order to better balance the mesh angles compared to version 3.6 (Figure 7). When the central cube is not inflated, some elements can have a poor skewness and/or poor aspect ratio in the vicinity of the central cube. Inflating it significantly improves both the

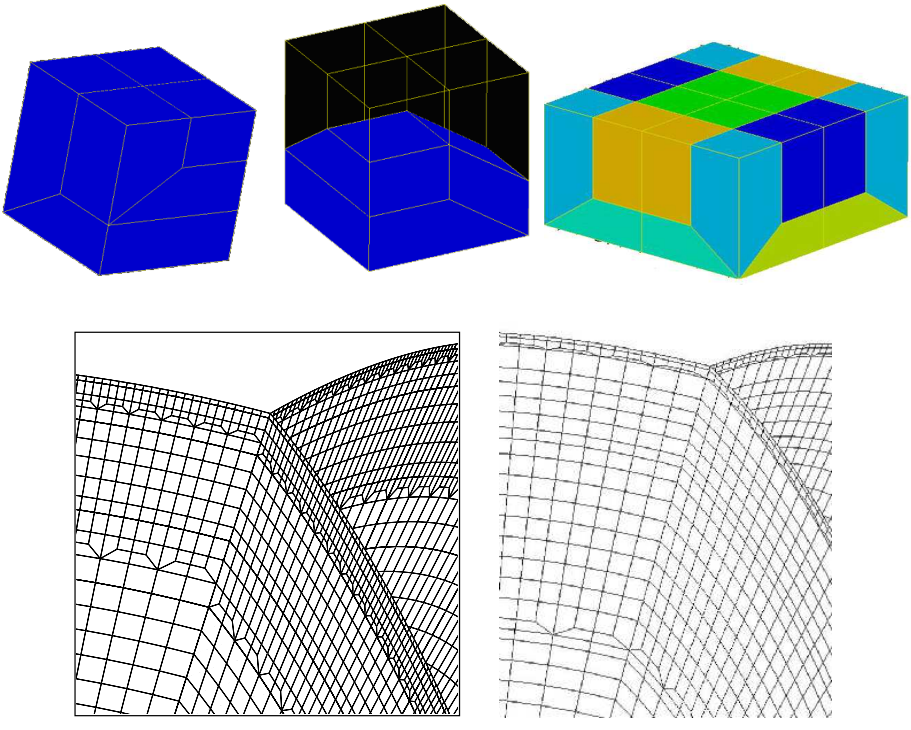


Fig. 6. In version 4.0 of SPECFEM3D, the mesh doubling of Figure 4 is accomplished in only one level instead of two in each mesh chunk and therefore three in the whole sphere, based on a more efficient geometrical 'doubling brick' (top, left and center) which is assembled in a symmetric block of four 'doubling bricks' based on mirror symmetry (top right). As a result, when we zoom on a region of contact between three of the six mesh chunks of Figure 2, we can see that while in the old mesh of version 3.6 (bottom left) there are three types of mesh chunks (labeled A, B and C in Figure 4), in version 4.0 of the code (bottom right) this poor property of the mesh has been suppressed and all the mesh chunks have the same shape and exact same number of elements, thus resulting in perfect geometrical mesh balancing.

skewness and the aspect ratio (both the average value for all the elements and the worst value for the most distorted element).

In the SEM one needs to assemble internal force contributions between neighboring slices, as mentioned above and in Figure 3. The pattern of communications needed to assemble such slices on the edges and corners of the six blocks of the cubed-sphere mesh can be determined from Figure 2 (for instance the valence of most surface points is 4, but it is three at the corners of the six blocks). Because the mass matrix is exactly diagonal, processors spend most of their time performing actual computations, and the amount of communications is comparatively small (in spite of the fact that the number of points to exchange increases approximately as N^2, but the polynomial degree N is always chosen

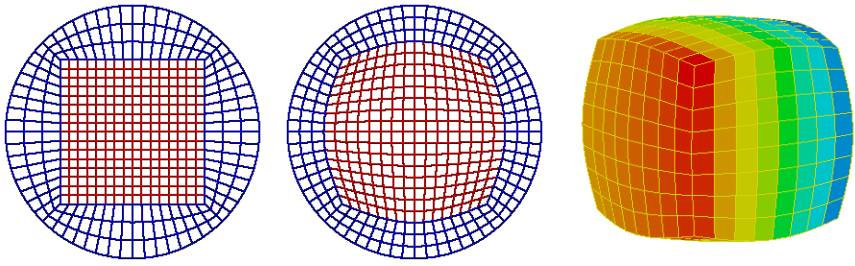


Fig. 7. In order to have more uniform mesh sampling in the inner core of the Earth, in version 4.0 of SPECFEM (middle) we slightly inflate the central cube in order to better balance the mesh angles compared to version 3.6 (left), as illustrated here in a 2D cut plane. In 3D this results in the inflated cube represented on the right.

small, between 4 and 7 in practice, see e.g. [3,8]). We thought about switching from the blocking MPI implementation used in version 3.6 to a non-blocking implementation in order to overlap the communications with calculations. This would imply first looping on the elements that are located on the edges of the mesh slices, computing their contributions, starting non-blocking SENDs of their contributions, and computing the rest of the elements inside the slices while the communications are being performed (see e.g. [12]). We implemented this strategy in a 2D version of our code (for simplicity) but did not notice any significant gain in terms of performance because the overall cost of communications is very small ($< 5\%$) compared to CPU time. We therefore concluded that there was no real need to switch to non-blocking MPI in the 3D version of the code.

3.6 Performance and Scaling Results

Let us perform an analysis of the improved code on 96 processors using ParaVer. Figure 5 shows that in the initial version 3.6, the number of L2 cache misses was very different between mesh slices, thus inducing severe load imbalance. In the improved version 4.0, L2 cache misses have been drastically reduced and very well balanced. The number of instructions executed is also very well balanced. As a result, useful duration of the calculations is well balanced too. In total, we gain a huge factor of 3.3 in terms of wall-clock time between both versions. This shows that the IBM PowerPC 970 is very sensitive to cache misses because the same run performed on an Intel Itanium and also on an AMD Opteron cluster shows a factor of 'only' 1.55 to 1.60.

Let us now analyze scaling by measuring wall-clock time per time step (averaged over 700 time steps in order for the measurement to be reliable) for a medium-size run performed on 24, 54, 96 and 216 processors (we also tried to run the test on 6 processors but it was too big to fit in memory). In Figure 8 we compare the scaling curve to the theoretical curve corresponding to perfect linear scaling, which we compute using the time measured on 96 processors and scaling it by the ratio of the number of processors used. The conclusion

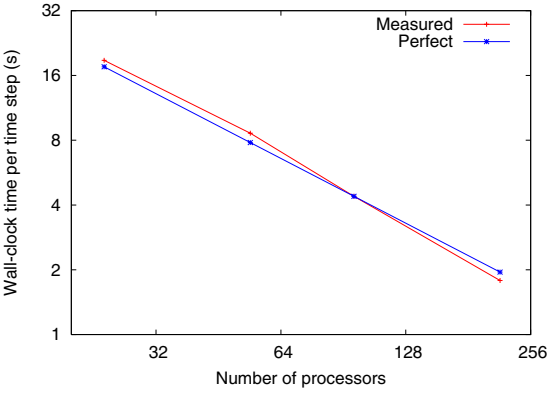


Fig. 8. Scaling in logarithmic scale measured (in red) for the same run performed on 24, 54, 96 and 216 processors, compared to perfect linear scaling (in blue) computed using the run on 96 processors as a reference. The two curves are very similar.

is that the scaling of the code is excellent. We therefore conclude that we are now ready to run the code for a real application on a very large number of processors of MareNostrum. MareNostrum has 2560 two-biprocessor blades, for a total of 10240 processor cores. Each blade has 8 gigabytes of memory, for a total of 20480 gigabytes of memory. For the final high-resolution run, we used 2166 processor cores and computed 50600 time steps of the explicit time integration scheme of the SEM algorithm. Total memory used was 2.5 terabytes. The code performed well and performance levels obtained were very satisfactory, the whole run took slightly less than 60 hours of wall-clock time (being the only user running on the corresponding dedicated blades). The geophysical analysis of the seismograms is currently under way.

4 Conclusions

MareNostrum has allowed us to reach unprecedented resolution for the simulation of seismic wave propagation resulting from an earthquake in the 3D inner core of the Earth using a spectral-element method implemented based upon MPI. A combination of better mesh design and improved point numbering has allowed us to balance the number of instructions very well, drastically reduce the number of L2 cache misses and also balance them very well, and as a result reach very good balancing in terms of the useful duration of the calculations in each mesh slice. BLAS3 or non-blocking MPI have not been required to achieve this, but using Altivec vector instructions such as multiply-and-add has allowed us to gain 20% in terms of CPU time.

Acknowledgments. The authors thank three anonymous reviewers for comments that improved the manuscript, and Jean Roman, Rogeli Grima, Nicolas

Le Goff, Roland Martin, Jean-Paul Ampuero and Jérémie Gaidamour for fruitful discussion. The idea of computing PKP seismic phases came from discussions with Sébastien Chevrot. The help of Sergi Girona, David Vicente, Judit Giménez and the BSC support group was invaluable to perform the calculations. This material is based in part upon research supported by HPC-Europa, European FP6 MIRG-CT-2005-017461 and French ANR-05-CIGC-002 NUMASIS.

References

1. Komatitsch, D., Vilotte, J.P.: The spectral-element method: an efficient tool to simulate the seismic response of 2D and 3D geological structures. Bull. Seismol. Soc. Am. 88(2), 368–392 (1998)
2. Seriani, G.: 3-D large-scale wave propagation modeling by a spectral element method on a Cray T3E multiprocessor. Comput. Methods Appl. Mech. Engrg. 164, 235–247 (1998)
3. Komatitsch, D., Tromp, J.: Introduction to the spectral-element method for 3-D seismic wave propagation. Geophys. J. Int. 139(3), 806–822 (1999)
4. Komatitsch, D., Tromp, J.: Spectral-element simulations of global seismic wave propagation-I. Validation. Geophys. J. Int. 149(2), 390–412 (2002)
5. Chaljub, E., Capdeville, Y., Vilotte, J.P.: Solving elastodynamics in a fluid-solid heterogeneous sphere: a parallel spectral-element approximation on non-conforming grids. J. Comput. Phys. 187(2), 457–491 (2003)
6. Komatitsch, D., Tsuboi, S., Ji, C., Tromp, J.: A 14.6 billion degrees of freedom, 5 teraflops, 2.5 terabyte earthquake simulation on the Earth Simulator. In: Proceedings of the ACM/IEEE Supercomputing SC 2003 conference, CD-ROM (2003), `www.sc-conference.org/sc2003`
7. Hughes, T.J.R.: The finite element method, linear static and dynamic finite element analysis. Prentice-Hall International, Englewood Cliffs (1987)
8. De Basabe, J.D., Sen, M.K.: Grid dispersion and stability criteria of some common finite-element methods for acoustic and elastic wave equations. Geophysics 72(6), T81–T95 (2007)
9. Sadourny, R.: Conservative finite-difference approximations of the primitive equations on quasi-uniform spherical grids. Monthly Weather Review 100, 136–144 (1972)
10. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the 24th National ACM Conference, pp. 157–172. ACM Press, New York (1969)
11. Jost, G., Jin, H., Labarta, J., Giménez, J., Caubet, J.: Performance analysis of multilevel parallel applications on shared memory architectures. In: Proceedings of the IPDPS 2003 International Parallel and Distributed Processing Symposium, Nice, France (April 2003)
12. Danielson, K.T., Namburu, R.R.: Nonlinear dynamic finite element analysis on parallel computers using Fortran90 and MPI. Advances in Engineering Software 29(3-6), 179–186 (1998)

Using a Global Parameter for Gaussian Affinity Matrices in Spectral Clustering

Sandrine Mouysset, Joseph Noailles, and Daniel Ruiz

IRIT-ENSEEIHT, University of Toulouse, France
sandrine.mouysset@enseeiht.fr, jnoaille@enseeiht.fr,
daniel.ruiz@enseeiht.fr

Abstract. Clustering aims to partition a data set by bringing together similar elements in subsets. Spectral clustering consists in selecting dominant eigenvectors of a matrix called affinity matrix in order to define a low-dimensional data space in which data points are easy to cluster. The key is to design a *good* affinity matrix. If we consider the usual *Gaussian affinity matrix*, it depends on a scaling parameter which is difficult to select. Our goal is to grasp the influence of this parameter and to propose an expression with a reasonable computational cost.

1 Introduction

Clustering has many applications in a large variety of fields : biology, information retrieval, image segmentation, etc. Spectral clustering methods use eigenvalues and eigenvectors of a matrix, called affinity matrix, which is built from the raw data. The idea is to cluster data points in a low-dimensional space described by a small number of these eigenvectors. By far, it is commonly agreed that the design and normalization of this affinity matrix is the most critical part in the clustering process. We are concerned with the Gaussian affinity matrices because they are very largely used. The expression of the Gaussian affinity matrix depends on a parameter σ and the quality of the results drastically depends on the good choice of this parameter. As said by several authors [3],[6] and [4], the scaling parameter controls the similarity between data. We propose a new expression based on a geometrical interpretation which is a trade-off between computational cost and efficiency and test it with classical challenging problems. This definition integrates both dimension and density of data.

2 Algorithm *Ng, Jordan and Weiss* (NJW)

Let $x_1, ..x_m$ be a m points data set in a n-dimensional euclidean space. The aim is to cluster those m points in k clusters in order to have better within-cluster affinities and weaker affinities across clusters. We suppose that the number k of targeted clusters is given. The affinity between two points x_i and x_j could

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 378–390, 2008.

be defined by $A_{ij} = exp(-\|x_i - x_j\|^2/\sigma^2)$ where $\|.\|$ is the euclidean norm. We consider the spectral clustering algorithm proposed by *NJW* [3] which is based on the extraction of dominant eigenvalues and their corresponding eigenvectors from the normalized affinity matrix A. This approach resumes in the following steps :

- Form the affinity matrix $A \in \mathbb{R}^{m \times m}$ defined by:

$$A_{ij} = \begin{cases} exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2}) \text{ if } i \neq j, \\ 0 \text{ otherwise} \end{cases}$$

- Construct the normalized matrix : $L = D^{-1/2}AD^{-1/2}$ with $D_{i,i} = \sum_{j=1}^{m} A_{ij}$
- Construct the matrix $X = [x_1 x_2 .. x_k] \in \mathbb{R}^{m \times k}$ by stacking the eigenvectors associated with the k largest eigenvalues of L
- Form the matrix Y by normalizing each rows in the $m \times k$ matrix X

- Treat each row of Y as a point in $\mathbb{R}^k$, and group them in k clusters via the *K-means* method
- Assign the original point x_i to cluster j if and only if row i of matrix Y was assigned to cluster j.

NJW justify this algorithm by considering an ideal case with three well-separated clusters. With the assumption that the points are already indexed by clusters consecutively, the affinity matrix has a block-diagonal structure. Thus, the largest eigenvalue of the normalized affinity matrix is 1, with multiplicity of order 3. The normalized rows of the extracted dominant eigenvectors are piecewise constant. In the field of the rows of these largest eigenvectors, it is easy to identify the three well-separated points that correspond to these three piecewise constant eigenvectors, and then to define the clusters accordingly. As already said in [4], one crucial step is to select appropriately the parameter σ and, in that respect, we have to decide between a *global* parameter as in [3], [2] and [1] or a *local* parameter that depends on the points x_i and x_j as in [6].

3 Towards the Choice of a Global Parameter from a Geometric Point of View

As already said in introduction, the purpose is to build an affinity matrix that can integrate both the dimension of the problem as well as the density of points in the given n-th dimensional data set.

For the sake of efficiency, we shall investigate global parameters that can be used to derive the affinity matrix in the usual way, as a function of the distances between points in the data set. To this end, we first make the assumption that the n-th dimensional data set is isotropic enough, in the sense that there does not exist some privileged directions with very different magnitudes in the distances

between points along these directions. Let us denote by $S = \{x_i,\, 1 \leq i \leq m\}$ the data set of points, and by

$$D_{\max} = \max_{1 \leq i,j \leq m} \|x_i - x_j\|\,,$$

the largest distance between all pairs of points in S. Under this first hypothesis, we can then state that the data set of points in essentially included in a n-th dimensional box with edge size bounded by $D_{\max}$.

If we expect to be able to identify some clusters within the set of points S, we must depart from the uniform distribution of m points in this enclosing n-th dimensional box. This uniform distribution is reached when dividing the box in m smaller boxes all of the same size, each with a volume of order $D_{\max}^n/m$, with a corresponding edge size that we shall denote as

$$\sigma = \frac{D_{\max}}{m^{\frac{1}{n}}} \tag{1}$$

What can be expected, indeed, is that if there exists some clusters, there must be at least some points that will be at a distance lower than a fraction of this edge size σ. Otherwise, the points should all be at a distance of order σ of each other, since we have made the assumption of isotropy and since all the points are included in the box of edge size $D_{\max}$.

Our proposal is thus to build the affinity matrix as a function of the ratio of the distances between points and the reference distance value $\frac{\sigma}{2}$. To incorporate the dimension n of the problem of clustering, we also propose to consider the control volumes around points instead of the square of the distances as commonly used. If we consider for instance the usual affinity matrix made of the exponential of these distances, we then propose to build the following matrix

$$A_{ij} = \{\exp\left(\frac{-\|x_i - x_j\|_2}{(\sigma/2)}\right)^n\}\,, \tag{2}$$

where $1 \leq i \leq m$ correspond to the row indexes and $1 \leq j \leq m$ to the column indexes in A, and to zero the diagonal in the usual way to get the affinity matrix to be used in the spectral embedding technique.

We first point out that this model relies upon the fact that the n-th dimensional box can be divided into smaller bricks in all directions. In other words, this means that the value $m^{\frac{1}{n}}$ is close to some integer and at least larger than 2. We shall come back later on this point, which will take some importance when the dimension n of the problem becomes large, in which case the above model might be weakened of slight modifications. This will be addressed in more details in the experiments.

Under the hypothesis that the n-dimensional data set is still isotropic enough, but when there exists some directions with varying amplitudes in the data, we can adapt slightly the computation of σ by considering that the set of points is included in a rectangular n-dimensional box. To approximate the volume of this non square box, we compute the largest distances between all pairs of points along each direction to define the size of the edges :

$$\rho_k = \max_{1 \le i \le n} x_{ik} - \min_{1 \le j \le n} x_{jk}, k \in \{1, \ldots, m\}\,.$$

The vector ρ incorporates the sizes of the intervals in which each variable is included separately and, in this case, we shall consider that the enclosing rectangular box has the same aspect ratio as the one defined by the intervals lengths given in ρ, and with maximum edge size given by $D_{\max}$. Then, we can take

$$\sigma = \frac{D_{\max}\sqrt{n}}{\|\rho\|_2} \left(\frac{\prod_{i=1}^{n} \rho_i}{m} \right)^{\frac{1}{n}}\,, \tag{3}$$

which resumes to equation (1) when ρ is all constant and the box is square. A more general way, which is the basis of the Mahalanobis distance, would be to compute the spectral orientation of the dispersion of the data to fix the axes and compute the amplitudes along these axes. But this is more computationally demanding and assumes that the original data are linked together in some particular way. In this case, we can expect some preprocessing must be done to prepare the data appropriately.

4 Measures of Clustering

Ng and Weiss [3] suggest to make many tests with several values of σ and to select the ones with least distortion in the resulting spectral embedding. In some cases, the choice of σ is not very sensitive and good results can be obtained easily. Still, there exists many examples where this choice is rather tight, as for example in cases with geometrical figures plus background noise. In the following of this section, we introduce two *measures of quality* that can be used to identify the interval of appropriate values for the choice of σ.

4.1 Ratio of Frobenius Norms

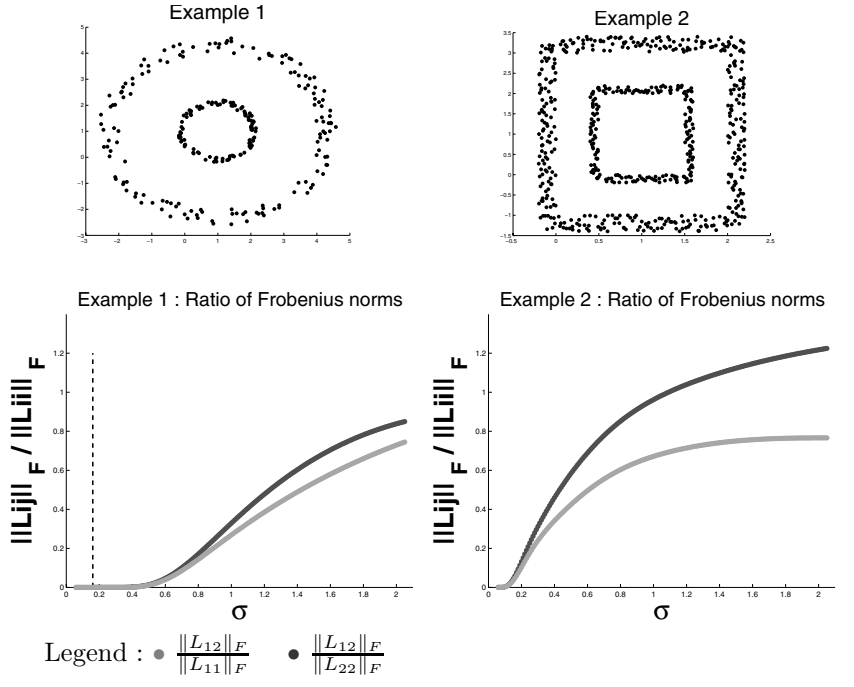
In general cases, the off-diagonal blocks in the normalized affinity matrix L are all non-zero and, for example, with $k = 3$, we can write :

$$\hat{L} = \begin{bmatrix} L^{(11)} & L^{(12)} & L^{(13)} \\ L^{(21)} & L^{(22)} & L^{(23)} \\ L^{(31)} & L^{(32)} & L^{(33)} \end{bmatrix}$$

We can then evaluate the ratios between the Frobenius norm of the off-diagonal-blocks and that of the diagonal ones.

$$r_{ij} = \frac{\|L^{(ij)}\|_F}{\|L^{(ii)}\|_F}$$

with $i \neq j$ and $i, j \in 1, .., k$ If the mean (or the max) of these values r_{ij} is close to 0, the affinity matrix has a near block diagonal structure. For example, in the following figure, we plot the value of these ratios in the case of two examples with two geometric clusters of points each.



From the behavior of these measures, we can see that there exists some interval in which the affinity matrix appears to be near block diagonal. This interval depends of course on the nature of the problem, and can be very different. For instance, in these examples, the length of this interval is of 0.4 in the first case and of 0.1 in the second one. The dash-dot line indicates the value of the heuristic (1) given in the previous section for the computation of σ. We can observe that this heuristic value falls in the corresponding intervals.

4.2 Confusion Matrix

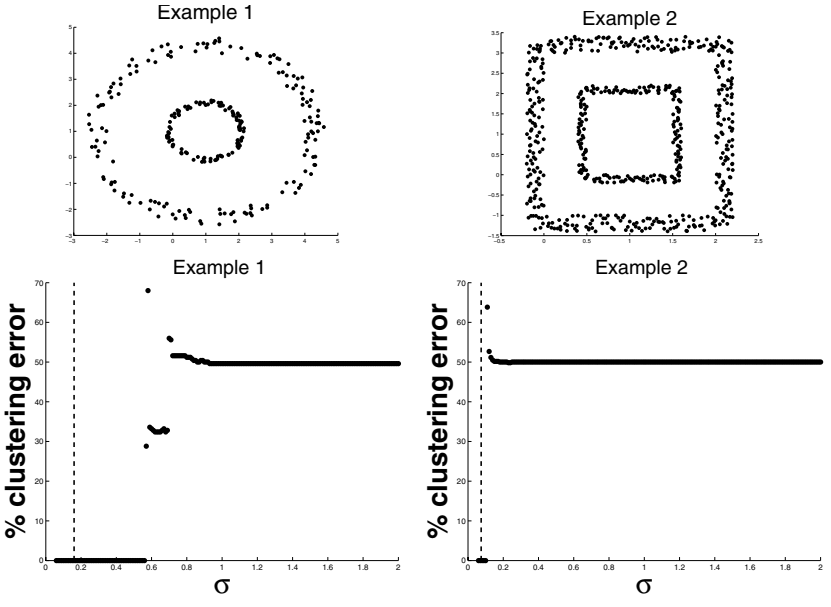
We now introduce an evaluation of the true error in clustering in the sense of the number of mis-assigned points within clusters. Let $C \in \mathcal{M}_{k,k}(\mathbb{R})$ be the so-called *confusion matrix* :

$$C = \begin{bmatrix} C^{(11)} & C^{(12)} & C^{(13)} \\ C^{(21)} & C^{(22)} & C^{(23)} \\ C^{(31)} & C^{(32)} & C^{(33)} \end{bmatrix}$$

(as for example in the case of three clusters) where $C^{(ij)}$ is the number of points that were assigned in cluster j instead of cluster i for $i \neq j$, and C^{ii} the number of well-assigned points for each cluster i.

We define the *percentage of mis-clustered points* by:

$$p = \frac{\sum_{i \neq j}^{k} C^{(ij)}}{m}$$



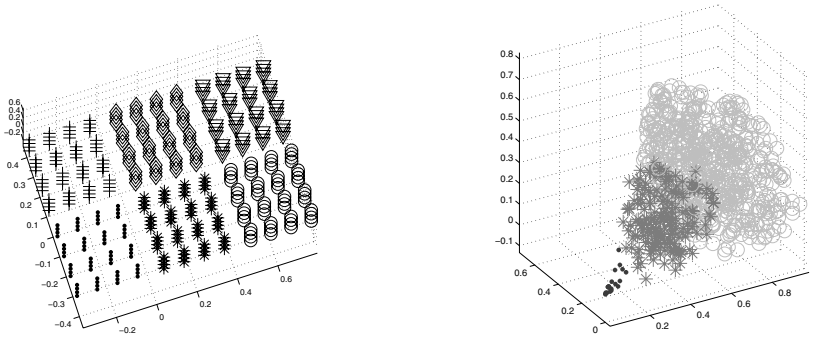
This matrix gives an estimate of the real error in the clustering method. The results from the previous two examples show that the interval value for the appropriate choice of parameter σ is approximatively the same as that observed with the ratios of Frobenius norms. We note that the clustering percentage of error varies almost instantaneously when σ just exits the appropriate interval. Again, we can observe that the value of the heuristic (1) corresponds to a value of σ with almost no clustering error.

5 Results

In order to validate the geometrical approach detailed in section 3, we consider two n-dimensional benchmark examples, one with six n-th dimensional uniform blocks slightly separated from each other, and another one made of pieces of n-spheres in $\mathbb{R}^n$ (see figure 1). The affinity matrix is defined by :

$$A_{ij} = \{\exp\left(\frac{-\|x_i - x_j\|_2}{\sigma/2}\right)^d\} \tag{4}$$

where d will be alternatively set to the different integer values from 1 to 5, in order to verify experimentally the adequacy of the power d (usually taken as $d = 2$) with respect to the dimension of the problem n, as suggested in section 3. To obtain the results given in the following tables, we have tried consecutive values of σ from 0.01 to 0.15 and computed the two error measures discussed in the previous section. This enabled us to determine approximately an interval of feasibility for the values of σ. The purpose of that was to verify if the heuristics (1) or (3) would belong to the appropriate intervals or not.



(a) n=3, m=384 (b) n=3,m=1200

Fig. 1. Example 1 & 2 : six blocks and three pieces of n-spheres

5.1 First Example: Six Blocks

This geometrical example is made of n-th dimensional blocks with uniform distribution each, slightly separated from each other, and is in perfect agreement with the assumption of isotropy used in the developments of section 3. Each block is composed of p^n points with a step size of 0.1 in each direction, and with $p = 4$ in the case of $n = \{2, 3\}$ and $p = 3$ in the case of $n = 4$. Finally, the blocks are separated from each other by a step size of 0.13. This example corresponds to the basic configuration that the heuristic (3) for σ should address well by default, and is therefore a fundamental case study.

In table (1), we indicate the results obtained for three different values of the dimension n of the problem, and we also indicate in each case the values σ_1 and σ_2 corresponding to the heuristics (1) and (3) respectively. For each of these dimensions, we vary the power d for the computation of the affinity matrix as indicated in (4), and we compute in all of these cases the intervals of feasibility for the values of σ with respect to the quality measures introduced in section 4. To determine these intervals in the case of ratio of Frobenius norms between blocks, the quality has been taken as acceptable when the mean of these ratios was inferior or equal to 0.15.

The results in table (1) show that the two heuristics σ_1 and σ_2 fall into the appropriate intervals in almost all cases. This is in agreement with the expectations in the sense that the affinity matrix is able to separate well the data. We also mention that the lengths of the interval, specially with the first quality measure, are larger for a value of d close to n, which is partly in favor of the consideration of the volumes instead of squared distances when building the affinity matrix in usual way.

5.2 Second Example: Three Pieces of n-Spheres with 1200 Points

This second example is built in the same spirit as the first one, except that each cluster has a different volume, and the spherical shape on some of the boundaries prevents k-means like techniques to separate well the clusters from scratch.

Table 1. 6 n-dimensional blocks

(a) $n = 2$ and $\sigma_1 = 0.1398$ and $\sigma_2 = 0.1328$

d	1	2	3	4	5
Ratio of Frobenius norm < 0.15	[0.02;0.3]	[0.02;0.56]	[0.02;0.6]	[0.02;0.62]	[0.02;0.6]
Clustering error	[0.12;1.6]	[0.06;1.24]	[0.08;1.06]	[0.1;1.18]	[0.12;1.1]

(b) $n = 3$ and $\sigma_1 = 0.1930$ and $\sigma_2 = 0.1510$

d	1	2	3	4	5
Ratio of Frobenius norm < 0.15	[0.02;0.3]	[0.02;0.58]	[0.02;0.64]	[0.02;0.66]	[0.02;0.66]
Clustering error	[0.02;3]	[0.06;2.2]	[0.04;1.4]	[0.04;1.2]	[0.04;1.2]

(c) $n = 4$, $\sigma_1 = 0.2234$ and $\sigma_2 = 0.1566$

d	1	2	3	4	5
Ratio of Frobenius norm < 0.15	[0.02;0.3]	[0.02;0.22]	[0.02;0.26]	[0.02;0.28]	[0.02;0.28]
Clustering error	[0.02;1.2]	[0.04;0.78]	[0.02;0.62]	[0.12;0.52]	[0.14;0.48]

As in the previous example, table (2) shows the results for the spectral clustering with the two quality measures in function of both d and n. We can observe again that the heuristics (1) and (3) are within the validity interval for both measures, and that for increasing values of the dimension n, the affinity matrix is better determined with the clusters when the power d is closer to n.

5.3 Image Segmentation

We consider now an example of image segmentation. In this case, we investigate two different approaches to define the affinity matrix :

- *as a 3-dimension rectangular box :* since the image data can be considered as isotropic enough because the steps between pixels and brightness are about the same magnitude, we can try to identify the image data as a 3-dimensional rectangular set and incorporate the heuristic (3) for σ in the affinity matrix given by (2).
- *as a product of a brightness similarity term and a spatial one :* the second possibility is to consider that the image data are composed of two distinct sets of variables, each one with specific amplitude and density. Indeed, the spatial distribution of the pixels is isotropic but the brightness is scattered into levels (256 maximum) and the brightness density cannot be derived from the number of points. Therefore, what is usually considered in papers dealing with image segmentation (see for instance [4,5]) is the product of an affinity matrix for the spatial data with an affinity matrix for the brightness values, each with its specific σ parameter reflecting the local densities. We then propose to build the affinity matrix in the following way :

Table 2. 3 pieces of n-sphere

(a) $n = 2$ and $\sigma_1 = 0.0288$ and $\sigma_2 = 0.0278$

d	1	2	3	4	5
Ratio of Frobenius norm < 0.15	[0.04;0.14]	[0.04;0.18]	[0.6;0.2]	[0.06;0.2]	[0.06;0.18]
Clustering error	[0.04;0.08]	[0.02;0.14]	[0.06;0.18]	[0.6;0.18]	[0.06;0.18]

(b) $n = 3$ and $\sigma_1 = 0.1074$ and $\sigma_2 = 0.1044$

d	1	2	3	4	5
Ratio of Frobenius norm < 0.15	[0.02;0.12]	[0.02;0.3]	[0.04;0.5]	[0.06;0.5]	[0.08;0.5]
Clustering error	[0.02;0.12]	[0.04;0.16]	[0.04;0.16]	[0.08;0.18]	[0.08;0.18]

(c) $n = 4$, $\sigma_h = 0.1704$ and $\sigma_2 = 0.1658$

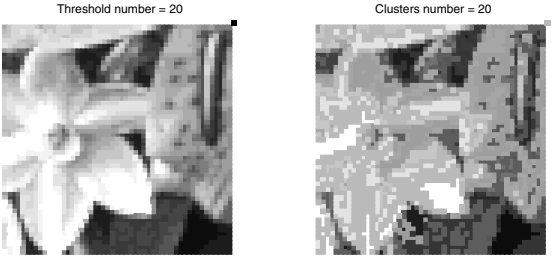
d	1	2	3	4	5
Ratio of Frobenius norm < 0.15	[0.04;0.04]	[0.02;0.04]	[0.04;0.08]	[0.06;0.1]	[0.1;0.16]
Clustering error	[0.02;0.02]	[0.04;0.1]	[0.06;0.16]	[0.08;0.18]	[0.1;0.2]

$$A_{ij} = exp(-\frac{\|x_i - x_j\|^2}{(\sigma_G/2)^2} - \frac{|I(i) - I(j)|}{(\sigma_B/2)}), \tag{5}$$

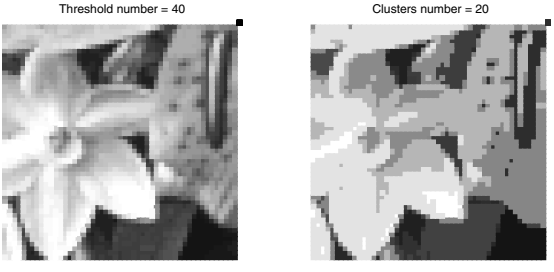
where $I(i)$ is the brightness value in $\mathbb{R}$ and x_i the coordinates of pixel i in $\mathbb{R}^2$. The parameter σ_G is given by (1) applied only to the spatial data, and σ_B is fixed to $(I_{\max}/\ell)$ with ℓ a characteristic number of brightness levels. For instance, in the following example, ℓ is equal to the number of threshold in the picture and σ_B will define the length of the intervals under which brightness values should be grouped together.

We point out that this way of doing is still in the spirit of the developments in section 3, because σ given by (1) reflects a clustering reference distance in the case of locally isotropic and scattered enough distribution of points. With 256 maximum brightness levels, the distribution cannot be considered anymore as locally scattered (lots of values are even equal to each other) and one must give a priori the characteristic distance under which brightness values can be clustered. We note also that the solution of taking $\sigma_B = I_{\max}/256$ would result in grouping the brightness values into clusters of length one approximately, and the segmentation of the image will require the analysis of a lot of clusters made of pixels close to each other and with about the same brightness level, equivalent to a very fine grain decomposition of the image.

In the following results, we test the approaches (2) and (5) for the computation of the affinity matrix on a 50×50 pixels picture. On the left, we show the original thresholded image and, on the right, the results obtained with either (2) in figure 2 or with (5) in figure 3.

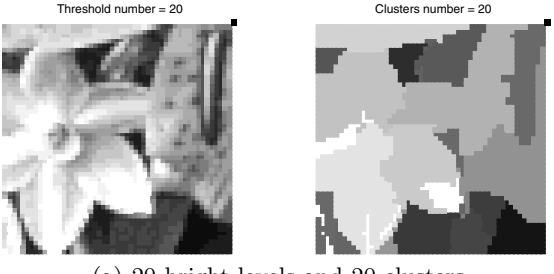


(a) 20 bright levels and 20 clusters

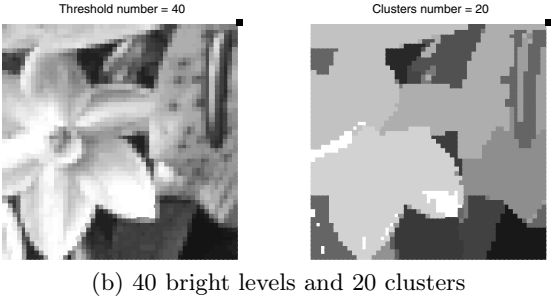


(b) 40 bright levels and 20 clusters

Fig. 2. Test of the 3-dimension rectangular affinity box on a flower

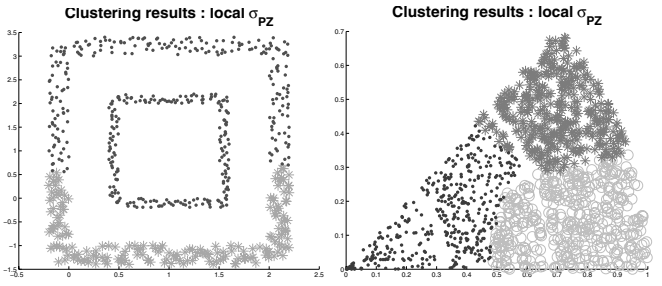


(a) 20 bright levels and 20 clusters



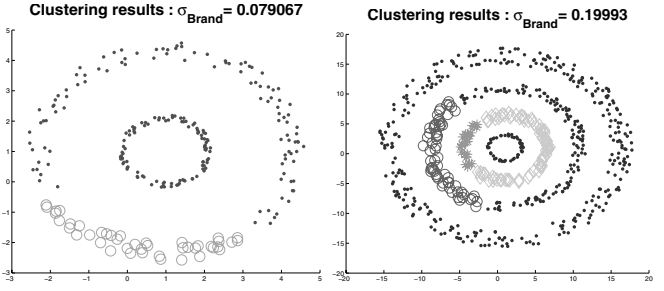
(b) 40 bright levels and 20 clusters

Fig. 3. Test of the product 2D by 1D affinity boxes on a flower



(a) $m = 600$ (b) $m = 1200$

Fig. 4. Examples with σ proposed by *Perona and al*



(a) $m = 250$ (b) $m = 650$

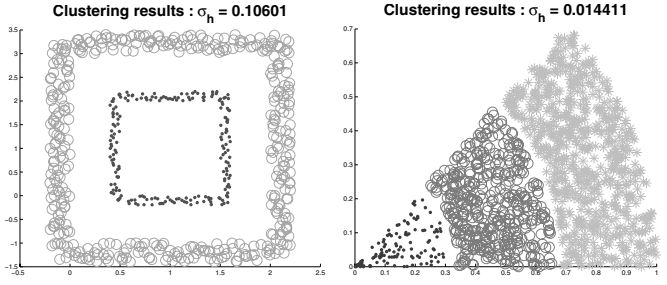
Fig. 5. Examples with σ proposed by *Brand and al*

In both cases, the results are visually acceptable. The 3-dimensional approach seems to provide nicer results than the 2D by 1D product, but we need more investigations to ensure which one of these two approaches is the best in general and to refine the results.

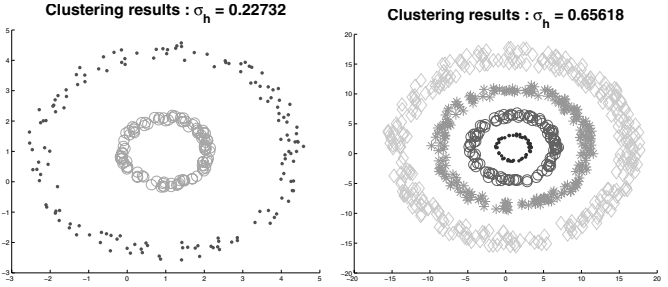
6 Remarks and Conclusions

The problematic of choosing an adequate parameter in order to improve the results has also been treated by some authors. Different points of view could be adopted.

– *Perona and Zelnik-Manor* [6] propose a locally approach. They assign a different scaling parameter σ_i to each point x_i in the data set. σ_i is equal to the distance between x_i and its P-th neighbors. This method gives great results in some kind of problems where the effect of local analyze provides enough information to create the clusters : for example, recovering a tight cluster within background noise. But computing a value of σ for each point x_i can be costly and the value P must be fixed empirically (P=7).



(a) $\sigma_h = 0.106$ (b) $\sigma_h = 0.014$



(c) $\sigma_h = 0.227$ (d) $\sigma_h = 0.656$

Fig. 6. Examples with the heuristic σ for $n = 2$

– *Brand and Huang* [2] define a global scale parameter : the mean between each data point and its first neighbor. In many examples, we obtain well clustered data representations.

In the examples introduced in figure 5, the density of points varies within each cluster. These results illustrate the fact that without global density information, it can be difficult to cluster well the data points in some cases. We test our definition (1) of global parameter for these examples in figure 6.

As recalled above, this global parameter gives good results in these four cases. We mention however that this heuristic parameter gives information about the spatial repartition of the data in a box of dimension $D_{\max}$. So when we have cases with an important noise density, the noise is difficult to separate from the existing clusters and can be assigned to its closest cluster. Only a local parameter can help to identify the noise from the cluster.

In conclusion, we have proposed a parameter for the construction of the affinity matrix used within spectral clustering techniques. This approach is adapted to n-dimensional cases, and based on a geometric point of view. With an isotropic assumption, this parameter represents the threshold of affinity between points within the same cluster. With quality measures such as ratio of Frobenius norms and confusion matrix, the rule of σ is observed and our definition is validated on

a few n-dimensional geometrical examples. We have also tried a case of image segmentation, but we still need deeper investigations and larger sets of test examples to ensure the validity as well as to determine the limitations of this approach. We plan also to test this general approach in a case of biologic topic.

References

1. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. Advances in Neural Information Processing Systems 14(3) (2002)
2. Brand, M., Huang, K.: A unifying theorem for spectral embedding and clustering. In: 9th International Conference on Artificial Intelligence and Statistics (2002)
3. Ng, A.Y., Jordan, M.I., Weiss, Y.: On spectral clustering: analysis and an algorithm. Proc. Adv. Neural Info. Processing Systems (2002)
4. Freeman, W.T., Perona, P.: A factorization approach to grouping. In: Burkhardt, H.-J., Neumann, B. (eds.) ECCV 1998. LNCS, vol. 1406. Springer, Heidelberg (1998)
5. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(8), 888–905 (2000)
6. Perona, P., Zelnik-Manor, L.: Self-tuning spectral clustering. In: NIPS (2004)

Comparing Some Methods and Preconditioners for Hydropower Plant Flow Simulations

Luiz M. Carvalho[1], Wagner Fortes[1], and Luc Giraud[2]

[1] Group of Environmental Studies of Hydropower Reservoirs (GESAR), Rio de Janeiro State University, R. Fonseca Teles, 121, 20550-013, Rio de Janeiro, RJ, Brazil
{luizmc,wfortes}@gmail.com
[2] ENSEEIHT-IRIT ,Toulouse, France
giraud@n7.fr

Abstract. We describe some features of a three-dimensional numerical simulator currently under development for studying water physico-chemical properties during the flooding of hydroelectric plants reservoirs. The work is sponsored by the Brazilian Electric Energy National Agency (ANEEL) and conducted with Furnas Centrais Elétricas S. A., the leading Brazilian power utility company. An overview of the simulator requirements is given. The mathematical model, the software modules, and engineering solutions are briefly discussed, including the finite element based transport module. We compare methods, iterative methods and preconditioners used to solve the sparse linear systems which arise from the discretization of three-dimensional partial differential equations.

1 Introduction

Is flooding of soils, consecutive to the creation of water reservoirs, a significant anthropic source of greenhouse gases (GHG) emissions? In a mid and long term perspective, can hydroelectrical energy be considered as a clean energy? The answers of the scientific and industrial communities to these questions are not conclusive [1], [2]. In order to participate in this discussion, a group of researchers has been developing a numerical simulator for studying water physico-chemical properties during the flooding of hydroelectric plants reservoirs [3, 4]. In the near future, this simulator will be able to analyze the production, stocking, consumption, transport, and emission of carbon dioxide (CO_2) and methane (CH_4) in reservoirs. The simulator comprises a Graphical User Interface (GUI) using OpenGL, and a Shell Interpreter. Geographical data in various formats are fed to the *Terrain* module, that generates the level sets and prepares the site geometry for the next module, *Phyto*. Drainage and phyto-physionomy data are added by *Phyto* and handed over to the *Mesh* module which generates the grid for the transport simulator. The prototype was developed with *Matlab* and is currently being rewritten in *C++*. The *Transport* module comprises the core of the simulator and uses a mixed finite element scheme.

The simulator is based on a nonlinear system of partial differential equations, the Navier-Stokes equations and a scalar transport equation [5, 6], presented below in a nondimensional form:

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 391–404, 2008.

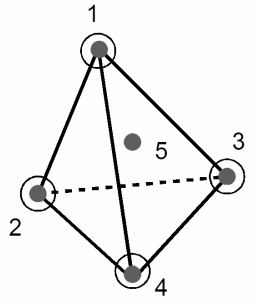


Fig. 1. Mini element with velocity in five points (centroid included) and pressure in four

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} &= -\nabla p + \frac{1}{\mathrm{Re}} \nabla \cdot \nu_t (\nabla \mathbf{u} + \nabla \mathbf{u}^T) \\ \nabla \cdot \mathbf{u} &= 0 \\ \frac{\partial c}{\partial t} + \mathbf{u} \cdot \nabla c &= \frac{1}{\mathrm{Re\ Sc}} \nabla \cdot D_t \nabla c + S \ , \end{aligned} \tag{1}$$

where $\mathbf{u}$, p are, respectively, the nondimensional velocity and pressure, *Re* is the Reynolds number, ν_t is the nondimensional effective viscosity, c is the advected scalar, Sc is the Schmidt number, D_t is the nondimensional effective diffusion tensor and S is a source term.

In the current version, the simulator implements a cubic tetrahedron element, mini element [7,8], with the velocity evaluated at the vertices and the centroid of the element; pressure is evaluated only in the vertices, see Figure 1.

For the treatment of these equations a common approach is the Galerkin method [9, 10] which transforms the equations (1) in a system of ordinary differential equations:

$$\begin{aligned} M\dot{\mathbf{u}} + \frac{1}{Re} K\mathbf{u} + Gp &= \hat{b}_1 \\ D\mathbf{u} &= b_2 \\ M_c \dot{c} + \frac{1}{Re\ Sc} K_c c &= b_3 \ , \end{aligned} \tag{2}$$

in equations (1) and (2) we are using the same symbols $\mathbf{u}$, p, c for continuous and discrete variables, and the matrices we describe in the next section.

A semi-Lagrangian method [11] is used for the time discretization. This approach changes the system (2) into:

$$\begin{aligned} M\left(\frac{\mathbf{u}_i^{n+1} - \mathbf{u}_d^n}{\Delta t}\right) + \frac{1}{Re} K\mathbf{u}^{n+1} + Gp^{n+1} &= \hat{b}_1 \\ D\mathbf{u}^{n+1} &= b_2 \\ M_c\left(\frac{c_i^{n+1} - c_d^n}{\Delta t}\right) + \frac{1}{Re\ Sc} K_c c^{n+1} &= b_3 \ . \end{aligned} \tag{3}$$

The variables from the previous time step are used to evaluate the chemical species field, uncoupling the hydrodynamics from the transport of the scalar variables. As the

simulator models three-dimensional space, the linear systems arising from these schemes are huge and it is mandatory to implement iterative methods for their solutions.

2 Coupled and Segregated Methods

The first two equations of (3) yield the following linear system:

$$\begin{pmatrix} A & G \\ D & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \qquad \text{or} \qquad \mathcal{A}x = b \,, \tag{4}$$

where $A = (\frac{1}{\Delta t}M + \frac{1}{Re}K) \in \mathbb{R}^{n\times n}$ is symmetric and positive definite, G, $D^T \in \mathbb{R}^{n\times m}$ and $n \geq m$, both have full rank. The unknown u represents the velocity and p the pressure in each point. Vectors $b_1 = (\hat{b}_1 + \frac{1}{\Delta t}M\mathbf{u}_d^n)$ and b_2 compose the constant right-hand side. The matrix $\mathcal{A}$ is known as saddle point matrix [12]. The third equation in (3) produces an easier system, that we are not treating in this work.

In order to uncouple the velocity and the pressure components in (4), one may use a segregated approach performing a block LU factorization of the original matrix:

$$\begin{pmatrix} A & G \\ D & 0 \end{pmatrix} \sim \begin{pmatrix} A & 0 \\ D & -D\tilde{A}^{-1}G \end{pmatrix} \begin{pmatrix} I & \tilde{A}^{-1}G \\ 0 & I \end{pmatrix} \,, \tag{5}$$

where $\tilde{A}$ is either equal to A or an approximation of A that is easier to solve. For the former, $S = -D\tilde{A}^{-1}G$ is the exact Schur complement matrix of the zero block of the matrix $\mathcal{A}$, and for the latter is an approximation of the Schur complement matrix. One has to compute:

Algorithm 1 (Projection Method)
Let v be an auxiliary variable.

1. $Av = b_1$;
2. $-D\tilde{A}^{-1}Gp = b_2 - Dv$;
3. $u = v - \tilde{A}^{-1}Gp$;

in a Fluid Dynamics framework, this alternative is called Projection method [13].

Depending on boundary conditions the complete system can be symmetric or non-symmetric. When A is symmetric and positive definite (SPD) and $G = D^T$, if $\tilde{A}$ is a diagonal matrix, then $-D\tilde{A}^{-1}G$ is also symmetric. In this case, both systems can be solved using the preconditioned conjugate gradient method (PCG) [14, 15]. When $G \neq D^T$ the approximated Schur complement matrix can be solved using GMRes [16] or BiCGStab [17]. The right-hand side of (5) replaces the original matrix involved in the linear system (4), and the computed solution accuracy depends on the quality of the approximation $\tilde{A}$ of A. Yet another segregated option is to obtain an approximation for the Schur complement matrix $-D\tilde{A}^{-1}G$ instead of an approximation for A. In this case, it is necessary to substitute the third step of Algorithm 1 by

$$Au = b_1 - Gp \,. \tag{6}$$

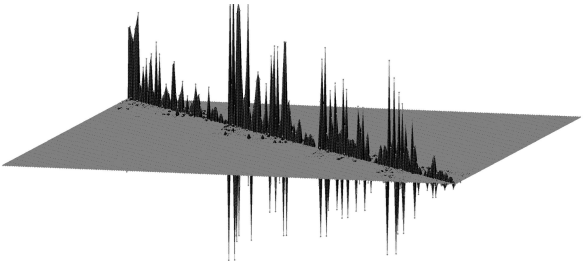


Fig. 2. A representation of a Schur complement matrix with tiny entries far from the main diagonal

The Schur complement matrix S, in our application, is dense and huge with order equal to the number of vertices of a three-dimensional mesh. However, Figure 2 represents a typical S matrix with tiny entries spread out of a few central diagonals. One may implement implicit and explicit alternatives for treating S. In the implicit case the matrix S is kept as the product $-D\tilde{A}^{-1}G$ and in the explicit alternative an approximation of S is computed. In section 3, we address some explicit alternatives we have tested.

The coupled method disregards the saddle point structure then equation (4) can be solved, for instance, by GMRes or BiCGStab. Nevertheless, this statement can be relaxed with the use of preconditioners that take into account the saddle point structure. The coupled approach spends much more computational resources than the segregated alternative but, in general, with a better numerical behavior, see section 6. Also, in order to obtain convergence it is necessary to use preconditioners, and reorderings.

3 Explicit Schur Complement Matrix

We have implemented three alternatives in order to assemble $S = -D\tilde{A}^{-1}G$: diagonal, probe, and complete approximation matrix.

3.1 Diagonal

$\tilde{A}$ is a diagonal matrix with two possibilities: diagonal and lumped. In the former, $\tilde{A} = \mathrm{diag}(A)$, the diagonal of A, in the latter

$$\tilde{A}_{ii} = \sum_{j=1}^{n} A_{ij},$$

where $\tilde{A}_{ii}$ is the diagonal element of $\tilde{A}$ in position (i, i). In both cases S is still sparse, as the product $\tilde{A}^{-1}G$ does not change the sparsity pattern of G, and the product of D by $\tilde{A}^{-1}G$, as long as the sparsity is considered, only puts in relation vertices that are "neighbors of neighbors", which is still quite sparse in the problem's three-dimensional mesh.

3.2 Probe

In [18], Chan and Mathew presented a quite simple idea for retrieving elements of an unassembled matrix E originated from a structured mesh. A set of probe vectors

composed by zeros and ones forms a rectangular matrix W, for instance, when using three probe vectors

$$W = \begin{pmatrix} 1\,0\,0 \\ 0\,1\,0 \\ 0\,0\,1 \\ 1\,0\,0 \\ 0\,1\,0 \\ \vdots\,\vdots\,\vdots \end{pmatrix}.$$

If E is a tridiagonal matrix the product EW retrieves exactly the three diagonals of E. If the diagonals of E far from the main diagonal have an important decreasing, this approach can provide a good approximation for E. Although the simulator mesh is not structured, we have noted, during the experiments, that S has tiny elements far from the main diagonal. So we have tested a probed approximation for S. As an approximation of the Cholesky factors of A are available, for instance, from a previous solution of the $(1,1)$-block using an incomplete Cholesky factorization of A, we have tested the following algorithm for obtaining an approximation of S:

Algorithm 2 (Probing S)
Let C and C^T be approximated Cholesky factors of A.

1. $F = GW$;
2. $H = C^{-1}(C^{-T}F)$, *forward and backward substitutions;*
3. $\tilde{S} = DH$;
4. $S = f(\tilde{S})$, *where f retrieves the desired entries.*

This alternative gave poor numerical results, we are not even presenting numerical results. Although we were aware of approaches tailored to saddle point problems with unstructured meshes [19], their implementation is a matter of future tests.

3.3 Complete Approximated Matrix

Another alternative is to use the following algorithm:

Algorithm 3 (Approximating S)
Let F has the same size as G.

1. $AF = G$, *multiple right-hand side problem;*
2. $\tilde{S} = -DF$;
3. $S = g(\tilde{S})$ *where g modifies the sparsity pattern of the operated matrix by applying a threshold.*

The first step is the most expensive one. Although it can be done in various ways, for instance as a multiple right-hand side problem, we have implemented it using the approximated Cholesky factors of A. In Matlab, this alternative can be quite time costly as one needs to exclude tiny elements in first and third steps out of the internal Matlab loops, but deserves further developments, mainly in C++, as the numerical results are comparable to other alternatives.

4 Segregated Solvers

After assembling the Schur complement matrix or an approximation of it (in the following we call both S), one has to solve the block system

$$\begin{pmatrix} A & 0 \\ D & S \end{pmatrix} \begin{pmatrix} I & \tilde{A}^{-1}G \\ 0 & I \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} b1 \\ b2 \end{pmatrix} \tag{7}$$

using Algorithm 1. As A is SPD, the PCG is the chosen algorithm for the first step of Algorithm 1; for solving the second step, for S, we have implemented GMRes.

As the actual solution for (4) was available, we could measure the error, instead of simply the residual, and we observed that, although cheaper than the coupled alternatives, the segregated solvers gave worse solutions. So we also tested an iterative refinement algorithm [20, 21] for the projection method by applying it to the residual of the computed solution. In our experiments three to five iterative steps were enough to improve the solution quality, see section 6.

5 Preconditioners

We describe preconditioners for both segregated and coupled methods. Based on [22], we also address preconditioners for the coupled approach by applying the segregated method to the residual. As the reorderings have an important impact on the global performance, for the sake of completeness, we also describe the used alternatives.

5.1 Preconditioners for Coupled Methods

We have tested seven preconditioners for the solution of $\mathcal{A}x = b$ by a coupled method using iterative Krylov subspace projection methods: GMRes and BiCGStab. For all tested cases, we approximate A by $\tilde{A}$.

Diagonal

1. Traditional, where $\tilde{A}$ is the diagonal of A.
2. Lumped, where $\tilde{A}$ is such that $\tilde{A}_{ii} = \sum_{j=1}^{n} A_{ij}$.

ILU(0). $\tilde{A}$ is such that its incomplete LU decomposition preserves the same sparse structure as A.

ILUT(τ). The ILUT(τ) is the incomplete LU with a threshold τ; this preconditioner is dynamically formed, as the elements smaller than τ in absolute value are discarded, so one does not know beforehand its sparsity pattern.

Segregated Preconditioner. Based on [22], we have used the segregated method, presented in section 4, as a preconditioner for the coupled method. The segregated methods are inexpensive, although, sometimes, have a poor numerical result, they can perform very well in practice as preconditioners .

Block Preconditioners. In [23] some preconditioners were proposed, we have implemented two of them: a block diagonal and a block triangular. The applications of these preconditioners generate inner-outer iterations schemes [24, 25, 26], and generate two linear systems that ought to be solved for the right-hand side residual.

Block Diagonal: This preconditioner uncouples the variables and writes

$$P_1 = \begin{pmatrix} A & 0 \\ 0 & DA^{-1}G \end{pmatrix} . \tag{8}$$

In exact arithmetic, the preconditioned matrix $P_1^{-1}\mathcal{A}$, as $\mathcal{A}$ is nonsingular, has three distinct eigenvalues (1, and $\frac{1\pm\sqrt{5}}{2}$), and the minimum polynomial has degree three, which guarantees the convergence of a Krylov subspace projection method in three iterations, in exact arithmetic.

Algorithm 4 (Block Diagonal)
1. $\mathbf{Az}_1 = \mathbf{r}_1$
2. $\mathbf{DA}^{-1}\mathbf{Gz}_2 = \mathbf{r}_2$

In the first step of Algorithm 4, we apply the PCG method with an incomplete Cholesky with threshold (ICCT) preconditioner. In the second step, we choose one of the available approximations to S, and we apply the GMRes method with an ILUT(τ) preconditioner.

Block Triangular: Another preconditioner based on the saddle point structure writes

$$P_2 = \begin{pmatrix} A & G \\ 0 & DA^{-1}G \end{pmatrix} . \tag{9}$$

The preconditioned matrix $P_2^{-1}\mathcal{A}$, as $\mathcal{A}$ is nonsingular, has two distinct eigenvalues: ± 1. In this case, the minimum polynomial has degree two, and a Krylov subspace projection method converges in two iterations, in exact arithmetic.

Algorithm 5 (Block Triangular)
1. $\mathbf{DA}^{-1}\mathbf{Gz}_2 = \mathbf{r}_2$
2. $\mathbf{Az}_1 = \mathbf{r}_1 - \mathbf{Gz}_2$

Here, In the first step, after choosing an approximation for S, we apply the GMRes method with ILUT(τ) as the preconditioner. Then we apply the PCG with ICCT preconditioner in the second step.

5.2 Segregated Preconditioners

We need two preconditioners in Algorithm 1 one for the first and third (depending on the construction of S) steps and other for the second step. We observe that the incomplete factorizations of A used as preconditioners in the first and third steps can be used for approximating S. Although the preconditioners are applied independently, we can interpret this as we have implemented the block diagonal version of [23], as addressed just above.

5.3 Reorderings

In addition, we consider the previous preconditioners combined with two block reorderings applied to (4). Namely, we investigate the column approximate minimum degree permutation (AMD) [27] and the symmetric reverse Cuthill-McKee (SRCM) [28, 29]. In order to preserve the saddle point structure, we reordered firstly the symmetric matrix A.

$$\hat{A} = P_A^T A P_A$$

where P_A is a column-permutation matrix for A. As the Schur complement matrix, S, was almost symmetric by structure, we also applied a symmetric reordering besides AMD.

$$\hat{S} = P_S^T S P_S$$

where P_S is a column-permutation matrix for S.

According to the given notations, $\hat{G}$ and $\hat{D}$ should be:

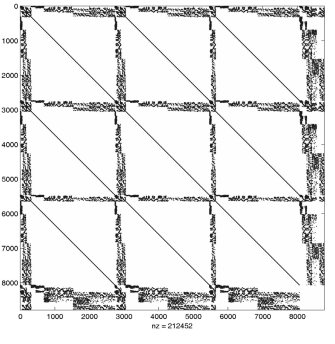
$$\hat{G} = P_A^T G P_S, \text{ and } \hat{D} = P_S^T D P_A$$

In Figure 3, we present the structure of a simulator typical saddle point matrix before and after reordering.

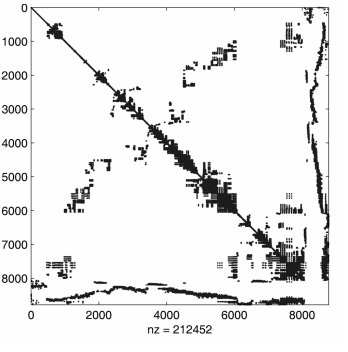
6 Numerical Tests

We have used Matlab 7, on a AMD X2 4200+ (dual core - 1024MB of cache), with 4Gb of RAM. The tested matrices are issued by the following three-dimensional problems:

1. a channel, the matrix order is 42,630 with 0.072% nonzeros,
2. a channel with step, the matrix order is 42,630 with 0.072% nonzeros,
3. a compartment of an actual reservoir, the matrix order is 34,578 with 0.085% nonzeros.



(a) Saddle point matrix without reordering.



(b) Saddle point matrix with AMD reordering.

Fig. 3. Examples of saddle point matrix before and after reordering

The problems were tested with CFL=1 and CFL=5, and Reynolds number of 10,000. The convergence criterion is Matlab's default: relative residual less than 10^{-6}. All the measures were done for one time step of a simulation when solving one linear system of the problems. The Krylov methods have 200 as the maximum number of iterations and GMRes was implemented without restart.

As long as reorderings are concerned, some remarks are necessary. Firstly, without any reordering the time for solving a typical problem is 80 times slower than with AMD or SRCM reorderings. As matter of fact, from a performance viewpoint, the reordering is the most important step in the reservoir problem simulation as soon as, as depicted in Figure 3(a), the arrowed structure of the matrix implies in a tremendous fill in process when computing a complete or an incomplete factorization. Secondly, the two ordering schemes are equivalent with a very light bias towards SRCM, so we are addressing figures only for this alternative.

The fill-in zero preconditioners, ILU(0) and ICC(0), did not converge or become singular for every test, for the three problems, so we are not presenting figures for these preconditioners. Also the probing construction for the Schur complement matrix, as we should have expected, presented a very bad behavior.

We present three types of tables: with complete preconditioners comparisons for the reservoir problem (Tables 1, 2, and 3), tables comparing the performance of the methods and preconditioners for the three problems with CFL=1 and CFL=5 (Tables 4 and 5), and tables comparing the segregated and coupled methods for the reservoir simulation (Tables 6 and 7).

For every table, the labels mean: **Preconditioner**, the preconditioner: None (without a preconditioner), Diagonal (classical diagonal), Lumped (lumped diagonal), ILUT(10^{-3}) (ILUT with an absolute threshold of 10^{-3}), Projection (the projection method used as a preconditioner), MGW1 (block diagonal described in section 5.1), and MGW2 (block triangular described in section 5.1); **Approx.** stands for the kind of Schur complement matrix approximation (**cam** is the complete approximated matrix), as described in section 3; **T.Prec** is related to the normalized time of the preconditioner

Table 1. Reservoir, coupled approach, left-preconditioned GMRes, CFL=5

Preconditioner	Approx. of S	T.Prec.	T.Sol.	Error	Iter.
None	$\cdots$		57.90	4.78	200
Diagonal	$\cdots$	1.00	4.80	0.92	200
Lumped	$\cdots$	1.00	4.81	0.99	200
ILUT(10^{-3})	$\cdots$	63.30	1.06	6.2e-7	9
Projection	diagonal	1.85	1.00	3.6e-5	13
	lumped	1.86	1.11	9.2e-5	12
	cam	42.30	2.33	7.2e-6	17
MGW1	diagonal	1.84	2.22	1.1e-4	27
	lumped	1.73	2.22	1.1e-4	27
	cam	42.30	3.81	6.7e-6	35
MGW2	diagonal	1.85	1.04	3.0e-5	13
	lumped	1.73	1.05	3.0e-5	13
	cam	42.30	1.95	9.9e-6	17

Table 2. Reservoir, coupled approach, right-preconditioned GMRes, CFL=5

Preconditioner	Approx. of S	T.Prec.	T.Sol.	Error	Iter.
Diagonal	$\cdots$	1.00	4.91	0.92	200
Lumped	$\cdots$	1.00	4.90	0.99	200
ILUT(10^{-3})	$\cdots$	63.30	1.00	6.2e-7	9
Projection	diagonal	1.85	1.66	1.9e-5	20
	lumped	1.86	1.40	8.4e-5	16
	cam	42.30	2.67	4.2e-7	20
MGW1	diagonal	1.85	3.30	5.4e-5	38
	lumped	1.73	3.30	5.4e-5	38
	cam	42.20	4.39	2.1e-6	40
MGW2	diagonal	1.85	1.56	6.5e-5	19
	lumped	1.73	1.56	6.5e-5	19
	cam	42.30	2.23	2.3e-6	20

Table 3. Reservoir, coupled approach, preconditioned BiCGStab, CFL=5

Preconditioner	Approx. of S	T.Prec.	T.Sol.	Error	Iter.
None	$\cdots$		1.24	0.92	190
Diagonal	$\cdots$	1.00	1.34	0.97	134
Lumped	$\cdots$	nc	nc	nc	nc
ILUT(10^{-3})	$\cdots$	63.30	1.00	1.1e-6	5
Projection	diagonal	1.85	23.39	4.6e-9	60
	lumped	1.86	27.39	1e-3.0	80
	cam	42.30	3.78	5.8e-8	14
MGW1	diagonal	1.85	27.30	7.0e-6	80
	lumped	1.73	27.30	7.0e-6	80
	cam	42.30	3.15	2.0e-7	14
MGW2	diagonal	1.85	27.30	7.0e-6	80
	lumped	1.68	27.30	7.0e-9	80
	cam	42.30	3.15	2.0e-7	14

Table 4. Comparing times in the three problems, with CFL=1

Problem	Solver	Preconditioner	Approx.	T.Prec.	T.Sol.	Error
Channel	RP-GMRes	Projection	diagonal	1.62	1.00	3.1e-8
	RP-GMRes	MGW 2	lumped	1.53	1.10	6.3e-8
Step	RP-GMRes	Projection	diagonal	1.62	1.02	9.9e-9
	RP-GMRes	MGW 2	lumped	1.53	1.11	1.2e-7
Reservoir	LP-GMRes	MGW 2	lumped	1.00	1.14	8.0e-6
	LP-GMRes	Projection	diagonal	1.07	1.13	4.6e-6

Table 5. Comparing times in the three problems, with CFL=5

Problem	Solver	Preconditioner	Approx.	T.Prec.	T.Sol.	Error
Channel	BiCGStab	Projection	lumped	1.00	1.00	1.4e-7
	RP-GMRes	Projection	diagonal	1.00	1.04	9.0e-8
Step	RP-GMRes	MGW 2	lumped	6.69	10.34	3.3e-8
	BiCGStab	MGW 1	lumped	6.98	10.43	2.9e-8
Reservoir	LP-GMRes	MGW 2	lumped	4.29	10.94	3.0e-5
	LP-GMRes	Projection	diagonal	4.59	10.34	3.6e-5

Table 6. Comparing segregated/coupled method for the reservoir problem with CFL=1

Method	Solver	Preconditioner	Approx.	T.Tot	Error
Segregated	Projection	MGW 1	lumped	1.00	1.1e-2
	IR		lumped	4.95	1.7e-4
Coupled	LP-GMRes	MGW 2	lumped	9.50	8.0e-6
	BiCGStab	Projection	diagonal	15.20	1.8e-9

Table 7. Comparing segregated/coupled method for the reservoir problem with CFL=5

Method	Solver	Preconditioner	Approx.	T.Tot	Error
Segregated	Projection	MGW 1	lumped	1.00	8.6e-2
	IR		lumped	2.36	2.2e-3
Coupled	LP-GMRes	MGW 2	lumped	4.51	3.0e-5
	BiCGStab	Projection	cam	28.60	5.8e-8

construction, **T.Sol** is the normalized time of the complete iterations of the preconditioned iterative Krylov methods, **Error** is the relative Euclidean norm of the actual error, i.e., the norm of the difference between the correct and the approximated solution divided by the correct solution norm. For computing the actual solution, we have used the backslash Matlab operator with iterative refinement in order to reach an accuracy of 10^{-15}.

In Tables 1 to 3, the column **Iter.** shows the number of iterations of the Krylov method. In Tables 4 and 5, **Problem** is the problem type - a channel, a channel with a step, or a reservoir branch. In Tables 4 to 7, **Solver** means the type of method, segregated or coupled. **IR** means the iterative refinement for the projection method. In Tables 6 and 7, **T.Tot** is the normalized total time for both the construction of the preconditioner and the iterative method, **Method** is the segregated or the coupled methods.

There are still some conventions. When the iterative method fails, we use **nc**. In the Approx. column, $\cdots$ means that there is no approximation for the Schur matrix, as we are treating, in this case, the coupled approach with a preconditioner that does not take into account the saddle point structure. All the times are normalized with respect to the least time in each column.

In Tables 1 to 3, we show figures for the reservoir problem, comparing different methods, Schur complement matrix approximations, and preconditioners using the same reordering scheme (SRCM) and CFL=5. We can observe that in Table 1 using

the coupled method with left-preconditioned GMRes, Projection and MGW2 present almost the same behavior, and ILUT(10^{-3}) although has a better numerical performance spends too much time for its construction. As we have mentioned before the Schur complement "cam" approximation is still too expensive in the construction step although with a good numerical result. In Table 2 using the coupled method with right-preconditioned GMRes, we can observe that almost the same behavior is found, with MGW2 and Projection alternatives performing similarly. Table 3, using BiCGStab for the same problem, shows that this iterative method does not perform well. Another remark is that the preconditioners that do not take into account the saddle point structure have a poor performance; in this case, ILUT(10^{-3}) although having a good numerical performance, has an expensive construction step.

In Tables 4 and 5, we rank the computational and the numerical behavior of all alternatives for the three studied problems. For each one, the first line presents the better result with respect to time, both preconditioner construction and iterative method execution times. The second line presents the least error amongst all experiments. The saddle point based preconditioners with quite simple approximations for the Schur complement matrix perform quite well. For simple problems, BiCGStab has the best behavior, outperforming GMRes.

Finally, in Tables 6 and 7, we address comparisons between the segregated and the coupled methods. For each one the first line presents the best normalized total time and the second line the least error. As we can observe in the last row of this table, the "cam" option has the least error, nonetheless with a quite expensive construction step. In the case of the segregated problem, we have implemented ICC(10^{-3}) for the (1,1)-block and ILUT(10^{-3}) for the Schur complement block. We can see that, depending on the accuracy of the solution, the segregated method with iterative refinement is quite competitive with the more expensive coupled approaches.

Acknowledgments. The first author would like to thank Norberto Mangiavacchi (Rio de Janeiro State University, UERJ) for fruitful discussions during the preparation of this work. The authors would like to thank the anonymous referees for their valuable comments and careful reading of the previous version of the paper. We are especially grateful to one of them for her/his suggestions that open new tracks for research that we intend to investigate in a future work.

References

1. Duchemin, E.: Hydroélectricité et gaz à effet de serre: évaluation des emissions et identification de processus biogéochimiques de production. PhD thesis, Université du Québec Montréal (April 2000)
2. Rosa, L.P., dos Santos, M.A.: Certainty and uncertainty in the science of greenhouse gas emissions from hydroelectric reservoirs. In: WCD Thematic Review Environmental Issues II.2, Cape Town, Secretariat of the World Commission on Dams, vol. II (November 2000)
3. Carvalho, L.M., Mangiavacchi, N., Soares, C.B.P., Rodrigues, W.F., Costa, V.S.: Comparison of preconditioners for the simulation of hydroeletric reservoir flooding. In: Proceedings of the 2007 International Conference On Preconditioning Techniques For Large Sparse Matrix Problems In Scientific And Industrial Applications (2007)

4. Mangiavacchi, N., Carvalho, L.M., Soares, C.B.P., Costa, V., Fortes, W.: Preconditioners for hydropower plant flow simulations. PAMM 7(1), 2100021–2100022 (2007)
5. Batchelor, G.K.: An introduction do fluid dynamics. Cambridge University Press, Cambridge (2000)
6. Lifshitz, E.M., Landau, L.D.: Fluid Mechanics, 2nd edn. Course of Theoretical Physics, vol. 6. Butterworth-Heinemann (1987)
7. Arnold, D.N., Brezzi, F., Fortin, M.: A stable finite element for the Stokes equations. Calcolo 21(4), 337–344 (1984)
8. Boffi, D., Brezzi, F., Demkowicz, L.F., Durn, R.G., Falk, R.S., Fortin, M.: Mixed Finite Elements, Compatibility Conditions, and Applications. Springer, Heidelberg (2008)
9. Quarteroni, A., Valli, A.: Numerical Approximation of Partial Differential Equations. Springer, Berlin (1994)
10. Grossmann, C., Roos, H.G., Stynes, M.: Numerical Treatment of Partial Differential Equations. Springer, Heidelberg (2007)
11. Durran, D.R.: Numerical Methods for Wave Equations in Geophysical Fluid Dynamics. Springer, Heidelberg (1998)
12. Benzi, M., Golub, G.H., Liesen, J.: Numerical solution of saddle point problems. Acta Numerica 14, 1–137 (2005)
13. Chorin, A.J.: Numerical solution of the Navier-Stokes equations. Mathematics of Computation 22, 745–762 (1968)
14. Hestenes, M., Stiefel, E.: Methods of conjugate gradients for solving linear systems. J. Res. Nat. Bur. Stand 49, 409–436 (1952)
15. Reid, J.K.: On the method of conjugate gradients for the solution of large sparse systems of linear equations. In: Reid, J.K. (ed.) Large Sparse Sets of Linear Equations, pp. 231–254. Academic Press, New York (1971)
16. Saad, Y., Schultz, M.H.: GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. 7(3), 856–869 (1986)
17. Van der Vorst, H.: BICGSTAB: a fast and smoothly converging variant of BI-CG for the solution of non-symmetric linear systems. SIAM Journal on Scien. and Stat. Computing 13(2), 631–644 (1992)
18. Chan, T.F., Mathew, T.: The interface probing technique in domain decomposition. SIAM J. Matrix Analysis and Applications 13 (1992)
19. Siefert, C., de Sturler, E.: Probing methods for saddle-point problems. ETNA 22, 163–183 (2006)
20. Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. SIAM, Philadelphia (2002)
21. Wilkinson, J.H.: Rounding Errors in Algebraic Processes. Notes on Applied Science No. 32, Her Majesty's Stationery Office, London. Prentice-Hall, Englewood Cliffs (1963); Reprinted by Dover, New York (1994)
22. Patankar, S.V.: Numerical heat transfer and fluid flow, p. 210. Hemisphere Publishing Corp., Washington (1980)
23. Murphy, M.F., Golub, G.H., Wathen, A.J.: A note on preconditioning for indefinite linear systems. SIAM Journal on Scientific Computing 21(6), 1969–1972 (2000)
24. Saad, Y.: A flexible inner-outer preconditioned GMRES algorithm. SIAM Journal on Scientific Computing 14(2), 461–469 (1993)
25. Simoncini, V., Szyld, D.B.: Theory of inexact Krylov subspace methods and applications to scientific computing. SIAM Journal on Scientific Computing 25(2), 454–477 (2003)

26. de Sturler, E.: Inner-outer methods with deflation for linear systems with multiple right-hand side. In: Proceedings of the Householder Symposium on Numerical Algebra, Pontresina, Switzerland, Householder Symposium XIII, pp. 193–196 (1996)
27. George, A., Liu, J.W.: The evolution of the minimum degree ordering algorithm. SIAM Review 31(1), 1–19 (1989)
28. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proc. 24th Nat. Conf., pp. 157–172. ACM, New York (1969)
29. George, A., Liu, J.: Computer solution of large sparse positive definite systems. Computational Mathematics. Prentice-Hall, Englewood Cliffs (1981)

Computational Models of the Human Body for Medical Image Analysis

Nicholas Ayache

Research Director
Asclepios Project-Team
INRIA
2004 Route des Lucioles,
06902, Sophia-Antipolis, France

Abstract. Medical image analysis brings about a collection of powerful new tools designed to better assist the clinical diagnosis and to model, simulate, and guide more efficiently the patient's therapy. A new discipline has emerged in computer science, closely related to others like computer vision, computer graphics, artificial intelligence and robotics.

In this talk, I describe the increasing role of computational models of anatomy and physiology to guide the interpretation of complex series of medical images, and illustrate my presentation with applications to cardiac and brain diseases. I conclude with some promising trends, including the analysis of in vivo confocal microscopic images.

N. Ayache, O. Clatz, H. Delingette, G. Malandain, X. Pennec, and M. Sermesant. Asclepios: a Research Project at INRIA for the Analysis and Simulation of Biomedical Images. In Proceedings of the Colloquium in Memory of Gilles Kahn, LNCS, 2007. Springer. Note: In press (20 pages).

This reference and others are available at
`http://www-sop.inria.fr/asclepios/`.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, p. 405, 2008.

Attaining High Performance in General-Purpose Computations on Current Graphics Processors

Francisco D. Igual, Rafael Mayo, and Enrique S. Quintana-Ortí

Depto. Ingeniería y Ciencia de los Computadores, Universidad Jaume I, 12.071–Castellón, Spain
{figual,mayo,quintana}@icc.uji.es

Abstract. The increase in performance of the last generations of graphics processors (GPUs) has made this class of hardware a coprocessing platform of remarkable success in certain types of operations. In this paper we evaluate the performance of linear algebra and image processing routines, both on classical and unified GPU architectures and traditional processors (CPUs). From this study, we gain insights on the properties that make an algorithm likely to deliver high performance on a GPU.

Keywords: Graphics processors (GPUs), general purpose computing on GPU, linear algebra, image processing, high performance.

1 Introduction

During the last years, since the emergence of the first generation of programmable graphics processors (GPUs), many studies have evaluated the performance of these architectures on a large number of applications. Thus, linear algebra operations [10,6], medical image processing [9,12], or database querying [8] are just a few examples of different arenas in which GPU computation has been successfully applied.

Recently, the design of GPUs with unified architecture and the development of general-purpose languages which enable the use of the GPU as a general-purpose coprocessor has renewed and increased the interest in this class of processors. Unfortunately, the rapid evolution of both the hardware and software (programming languages) of GPUs has outdated most of the performance studies available to date.

In this paper, we design and implement a reduced collection of "benchmark" routines, composed of four linear algebra operations (matrix-matrix product, matrix-vector product, saxpy, and scaling of a vector) and an image processing kernel (convolution filter). These routines are employed to evaluate the impact of the improvements introduced in the new generation of GPUs (*Nvidia G80*), comparing the results with those obtained on a GPU from a previous generation (*Nvidia NV44*) and current general-purpose processors (*AMD Athlon XP 2400+* and *Intel Core 2 Duo*). The ultimate purpose of this evaluation is to characterize

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 406–419, 2008.

the properties that need to be present in an algorithm so that it can be correctly and efficiently adapted into the GPU execution model.

The rest of the paper is organized as follows. Section 2 describes the basic architecture and execution model of both the old and new generations of GPUs Section 3 characterizes the routines in the benchmark collection. Sections 4 and 5 evaluate the performance of the benchmark routines on the *Nvidia NV44* and the *Nvidia G80*, respectively, comparing the results with those obtained on a CPU, and identifying a set of properties that must be present in an algorithm to deliver high performance on that GPUs. Finally, Section 6 summarizes the conclusions that can be extracted from our analysis.

2 GPU Architecture and Execution Model

2.1 GPU Graphics Pipeline

The graphics pipeline consists of a set of sequential stages, each one with a specific functionality and operating on an specific type of data. The process transforms original graphical information (vertices) into data suitable for being shown on display (pixels). Figure 1 illustrates the usual stages (or phases) that form the graphics pipeline.

Current GPUs implement this pipeline depending on the generation they belong to. Thus, classical GPUs have specific hardware units, known as *shaders* (or processors), for each one of the stages of the graphics pipeline. On the other hand, GPUs from the latest generation have a *unified shader* (or unified processor), with the ability to both execute any of the stages of the pipeline and work with any type of graphical data.

2.2 Classical Architecture

Until 2006 GPUs were based on a design where each pipeline stage was executed on a specific hardware unit or processor inside the pipeline. Thus, e.g., vertices are processed by *vertex processors* while pixels (also called fragments) are transformed by *fragment processors*. In practice, general-purpose algorithms implemented on these classical architectures exploit fragment processors only, due to their larger number and broader functionality. Fragment processors operate in SIMD mode, taking a fragment as input, and processing its attributes;

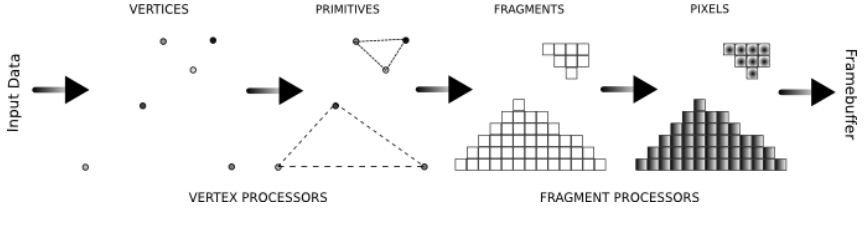


Fig. 1. Graphics pipeline process with its main stages

they can also process vectorial data types, working simultaneously on the four components of a fragment (R, G, B, and A). This class of hardware is able to read from random memory locations (commonly known as a *gather* operation in graphics algorithms), but can only modify one memory position per processed fragment, the one associated with the position of the fragment. This lack of support for *scatter* is one of the main restrictions of the classical GPU.

In the latter generations of this "classical architecture", programming capabilities were added to vertex and fragment processors. Altogether, the previous characteristics enable the use of fragment processors as a hardware platform to process non-graphical data. Unfortunately, the graphical-oriented design of this class of hardware, its SIMD execution model, the lack of a sophisticated memory hierarchy and the use of graphical-oriented APIs are problems for an efficient implementation of general-purpose applications on the GPU.

2.3 Unified Architecture

In 2006 a new generation of GPUs was introduced, with a completely different architectural design. These new platforms feature a *unified architecture*, with one processing unit or unified shader that is able to work with any kind of graphical data, transforming the sequential pipeline in Figure 1 into a cyclic one, in which the behavior of the unified shader varies depending on the stage of the pipeline that it is being executed at each moment.

There are several characteristics in the new generation of GPUs which specifically favor their use as a general-purpose coprocessor: in general, the clock frequency of the unified shader is much higher than that of a fragment processor (even though it is still much lower than the clock frequency of current CPUs); the shader consists of a large collection of computation units (up to 128, depending on the GPU version), called Streaming Processors (SPs), which operate in clusters of 16 processors in SIMD mode on the input data stream; and the architecture includes a sophisticated memory hierarchy, which comprises a L2 cache and small fast memories shared by all the SP in the same cluster.

These hardware advances are complemented with the CUDA [3] general-purpose programming library, which eases the programming effort on these platforms. In fact, CUDA has been proposed as a standard (although only compatible with Nvidia hardware) to program the new generation of GPUs, without the requirement of learning more complex graphics-oriented languages.

3 Benchmark Collection

In order to identify the algorithmic properties that yield correct and efficient codes for the GPU execution model, we have studied three major computational aspects of algorithms:

Data parallelism. The replication of functional units inside the GPU (fragment processors in the non-unified architectures, SPs in the unified architectures) makes this class of architectures specially appropriate for applications which exhibit a high degree of data parallelism.

Input data reutilization. The simple memory hierarchy in non-unified GPUs makes it difficult to exploit the locality of reference; in these architectures, high memory latency and limited bus bandwidth imply a penalty cost much higher than in a CPU; for this reason, input data reutilization is one of the biggest issues when trying to attain high performance on graphics processors.

Computational intensity per stream element. Due to the previous restriction, to achieve high performance the expensive cost of memory references should be masked with a high number of operations per memory access.

Our benchmark collection is composed of four *Basic Linear Algebra Subprograms* or BLAS [5]: the matrix-matrix product (`SGEMM`), the matrix-vector product (`SGEMV`), the "saxpy" (`SAXPY`), and the scaling of a vector (`SSCAL`); and a convolution filter, common in image processing. From the computational viewpoint, the routines in the benchmark present the following properties:

SGEMM. The matrix multiplication routine,

$$C = \alpha \cdot A \cdot B + \beta \cdot C$$

where A is $m \times k$, B is $k \times n$, and C is $m \times n$, being α and β scalars, features some properties that make it a good candidate to achieve good results when mapped into graphics hardware. It exhibits a regular memory access pattern, a high degree of data parallelism, and a very high computational load. On the other side, it is interesting to study the importance of the high input data reutilization in this type of algorithm.

For our study, we have chosen square matrices to evaluate the performance of the routine, and a non-transposed memory layout. The scalars α and β were set to 1. For a detailed study of the `SGEMM` performance on a GPU, refer to [1].

SGEMV. The matrix-vector multiplication routine

$$y = \alpha \cdot A \cdot x + \beta \cdot y$$

where A is a $m \times n$ matrix, x and y are vectors of length n and α and β are scalars, exhibits a smaller input data reutilization than `SGEMM`. Thus, while each input element for the `SGEMM` routine is used $O(n)$ times to compute the result, `SGEMV` reutilizes $O(n)$ times the data of the input vector, but only $O(1)$ times the data of the input matrix. This behavior makes the matrix-vector product routine a more streaming-oriented code, and so it is theoretically possible to achieve better results on a GPU. The simplest form of de `SGEMV` routine will be evaluated, with the matrix A not transposed in memory, and $\alpha = \beta = 1$.

SAXPY and SSCAL. The BLAS-1 routines `SAXPY` and `SSCAL`

$$y = \alpha \cdot x + y \qquad\qquad x = \alpha \cdot x$$

where x and y are vectors, and α is a scalar, are specially interesting for graphics processors, as they do not reutilize input data at all. These algorithms fit perfectly to the GPU architecture explained in Section 2. In fact, they can be seen as fully stream-oriented algorithms, where the input is a stream of data (for the `SSCAL` routine) or two streams (for the `SAXPY`), operations are performed over each of the elements of the input stream, without any kind of data reutilization, and finally an output data stream is returned.

The main difference between these two operations, from the performance viewpoint, is the amount of computational load per stream element. Thus, `SAXPY` performs twice as many operations as `SSCAL` per element. This difference offers some information on the importance of the computational load in the performance of the processor.

2D Convolution. Image processing algorithms traditionally exhibit a high performance when executed on graphics processors. More specifically, the convolution filters exhibit some properties which favor GPU hardware. First, the high degree of data parallelism will take advantage of fragment processors (or SP) replication of modern GPUs. Second, input data reutilization is very low (proportional to the size of the applied filter, usually small). Third, the computational load per calculated element is high, and based on multiply-and-add (MAD) operations, for which the GPU is specially appropriate.

For our evaluation purposes, we have implemented a bidimensional convolution filter with a square mask of different sizes, comparing optimized versions on CPU, using tuned BLAS libraries, and on GPU, using optimized Cg and CUDA implementations.

4 Previous Generation GPU-CPU Comparison

4.1 Experimental Setup

In this first experiment, we have chosen two experimental platforms of the same generation, an *AMD AthlonXP 2400+* CPU and a *Nvidia NV44* GPU processor (both from year 2004), so that we can do a fair comparison between general-purpose and graphics processors. Details on these architectures are given in Table 1. The GNU `gcc` 4.1.2 compiler is employed in the evaluation.

4.2 Implementation Details

The highly tuned implementation of linear algebra kernels in GotoBLAS 1.15 [7] was used to evaluate the performance of the CPU. The convolution filter implementation was built on top of GotoBLAS, using exclusively fully optimized BLAS operations.

On the other hand, the GPU was programmed using OpenGL and the Cg language (version 1.5). The routines were adapted to the architecture of the *Nvidia NV44* in order to optimize performance, as is briefly described next.

Table 1. Description of the hardware used in our first experimental study.

	CPU	GPU
Processor	*AMD AthlonXP 2400+*	*Nvidia GeForce 6200*
Codename	Thoroughbred A	NV44A
Clock frequency	2 GHz	350 MHz
Memory speed	2×133 MHz	2×250 MHz
Peak performance	8 GFLOPS	11.2 GFLOPS
Bus width	64 bits	64 bits
Max. bandwidth	2.1 GB/s	4 GB/s
Memory	512 MB DDR	128 MB DDR
Bus Type	AGP 8x (2 GB/s transfer rate)	
Year	2004	2004

For routine `SGEMM`, we start from a simple implementation, applying successive refinements in pursue of high performance. First, we adapt the original algorithm using the vectorial capabilities of the fragment processors, as proposed in [4]. This type of optimization usually yields a four-fold increase in performance, and is frequently applied to all types of GPU codes. In addition, we try to exploit the simple cache hierarchy of the *Nvidia NV44* by implementing a multipass algorithm, following the ideas in [11]. The goal here is analogous to blocking techniques for CPUs; however, this technique often delivers poorer results on GPUs as the multiple memory writes after each rendering pass penalize the global performance. In general, an SIMD architecture attains higher performance when the instructions are executed only once on the data stream.

We have also implemented optimized versions of routines `SGEMV`, `SAXPY`, and `SSCAL` which exploit the vectorial capabilities of the GPU by applying analogous optimizations to those described above for routine `SGEMM`.

Convolution filters allow us to introduce simple but powerful optimizations starting from a basic implementation. Our proposal to achieve high performance when executing this operation on a GPU is to divide the original $N \times N$ image into four $N/2 \times N/2$ quadrants. For simplicity, we assume here that N is a multiple of 2; the overlap applied to the boundaries is not illustrated. We then map the (i, j) elements of the four quadrants onto the four channels (R, G, B, and A) of an $N/2 \times N/2$ data structure. Since a GPU can process four-channel tuples as a scalar element, we can get up to four times higher performance with this type of optimization. Figure 2 illustrates the process. Although this strategy is quite simple, it illustrates the type of optimizations that can be easily applied when implementing general-purpose algorithms on a GPU.

4.3 Experimental Results

By analyzing the experimental results, the goal to determine which algorithmic properties (computational aspects in Section 3) favor the execution of an algorithm on a GPU with a classical architecture.

Data Reutilization: `SGEMM` vs. `SGEMV`

Figure 3 shows the results for routines `SGEMM` and `SGEMV` on the CPU and GPU. On the latter architecture, we report two different MFLOPs rates, labeled as "`GPU`"/

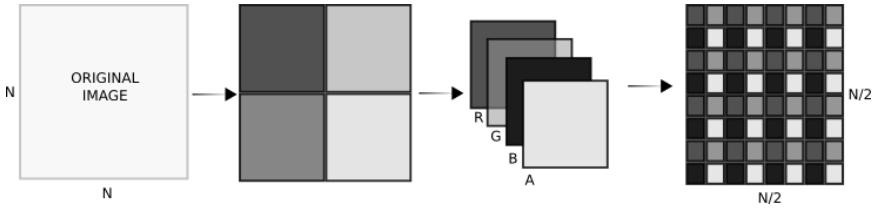


Fig. 2. Optimization applied to the computation of a convolution filters on a GPU with a classical architecture

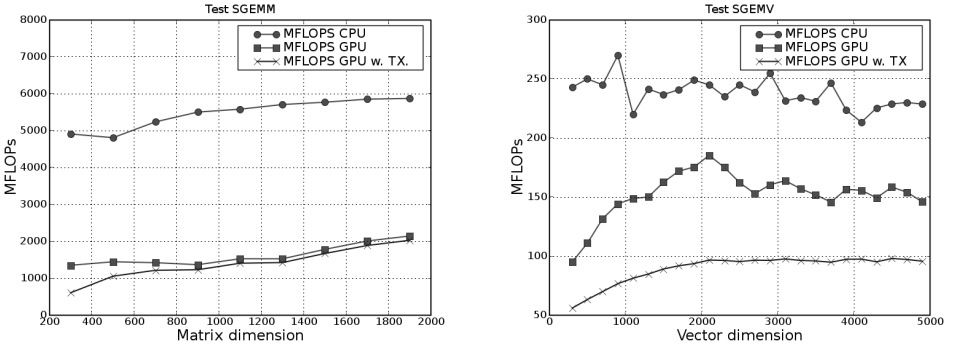


Fig. 3. Performance of routines `SGEMM` (left-hand side) and `SGEMV` (right-hand side) on the *AMD AthlonXP 2400+* CPU and the *Nvidia NV44* GPU

"`GPU w. TX`", obtained respectively by measuring only the execution time on the GPU or timing also the period required to transfer data and results between RAM and video memory. The high input data reutilization of the matrix-matrix product (see left-hand side plot) explains why the routine in Goto BLAS, which exploits the sophisticated cache hierarchy of the AMD CPU, outperforms the GPU implementation by a factor up to 4. The right-hand side plot illustrates how, when the data reutilization is lower as, e.g., in the matrix-vector product, the difference in performance between the CPU and GPU routines decreases, though still favors the CPU (between two and three times higher MFLOPs rate on this architecture).

The figure also reports that the impact of the data transference, however, is less important for routine `SGEMM`, which carries out a higher computational load per element that is transferred through the bus. From the previous results, it is possible to conclude that the amount of data reutilization is an important factor in order to achieve high performance on a GPU.

Computation Load Per Stream Element: BLAS-1 Routines

Therefore, one could expect that BLAS-1 operations (`SAXPY` and `SSCAL`) will deliver a high MFLOPs rate on this class of hardware. Surprisingly, as shown

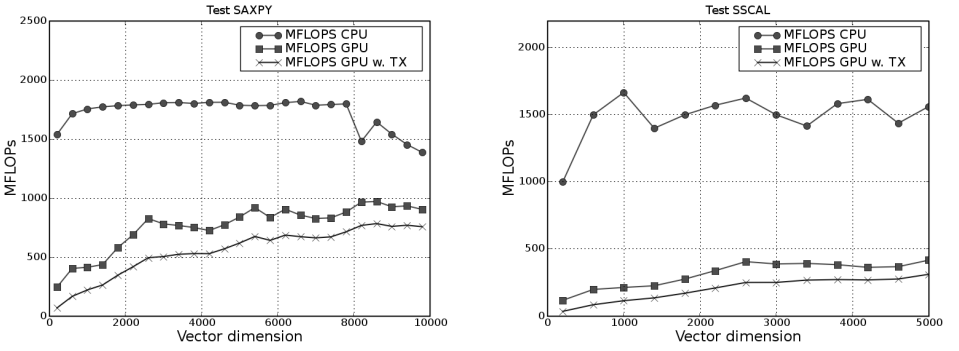


Fig. 4. Performance of routines `SAXPY` (left-hand side) and `SSCAL` (right-hand side) on the *AMD AthlonXP 2400+* CPU and the *Nvidia NV44* GPU

in Figure 4, we get a poor performance for our implementations of `SAXPY` and `SSCAL`, much lower than those of the corresponding CPU implementations.

This behavior can be explained as follows: the scarce amount of computational load per memory access in BLAS-1 operations limits their performance. This is partially due to the lower efficiency of the memory system of the *Nvidia NV44* GPU, with a poor use of cache memories. The elaborated cache memory of the CPU, and its efficient use by the optimized routines in Goto BLAS, are the reasons for such a notable difference in efficiency. Furthermore, results on the GPU are slightly better for `SAXPY` when compared with the corresponding implementation on CPU than for `SSCAL`, as the computational load per stream element calculated in the former operation is twice as high as that of `SSCAL`.

In conclusion, high computational load per stream element is one of the basic conditions for an algorithm to deliver high performance when executed on GPU.

Bidimensional Convolution Filters

Convolution filters combine in the same operation a set of very favorable properties for GPUs: high data parallelism, low input data reutilization, and high computational load per stream element. Figure 5 shows the results of the implementations of the convolution filter on the CPU and GPU. The optimized implementation on GPU (labeled as ‘‘`GPU4`’’) employs the four channels of each element of the input stream in order to store data (as explained at the end of Section 4), attaining a speed-up factor close to 4x with respect to a basic GPU implementation (labeled as ‘‘`GPU`’’). The comparison between this implementation and the optimized CPU version shows a comparable performance between the optimized CPU implementation and the optimized GPU one.

Convolution filters are the type of algorithms that better fit into the execution model of GPUs with classical architecture. These operations exhibit all the properties that make good use of GPUs computational power and, at the same time, hide those aspects in which CPUs are better than graphics processors (basically at memory access).

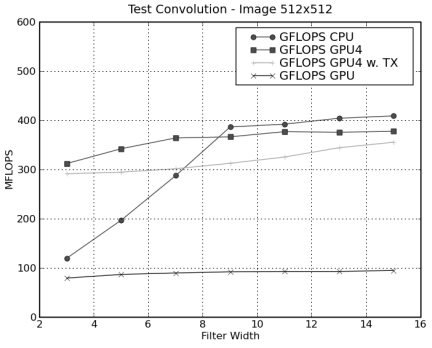


Fig. 5. Performance of the implementations of the convolution filter on the *AMD AthlonXP 2400+* CPU and the *Nvidia NV44* GPU

Impact of Data Transfers

From the empirical results, it is possible to conclude that the data transfer stage previous to any operation executed on GPU is a penalty to the final performance, although the overhead introduced is not critical. For this generation of GPUs, the AGP port is not a significative bottleneck for the overall computation process. In fact, the impact of data transfer is minimal for those routines in which the computational load per transferred element is high, e.g. matrix-matrix multiplication or convolution.

5 New Generation GPU-CPU Comparison

5.1 Comparison Goals

Although the study of the non-unified generation of GPUs has identified some of the characteristics desirable in algorithms that target GPUs with classical architecture, it is also interesting to carry over this study to new generation GPUs with unified architecture. The goal of this study is to verify if our previous insights also hold for these new architectures, and to evaluate how the hardware and software improvements (at computational power, memory hierarchies and interconnection buses level) affect the performance of the implemented routines.

5.2 Experimental Setup

In this second set of experiments, we again chose two experimental platforms from the same generation, an *Intel Core 2 Duo* CPU and a *Nvidia GeForce 8800 Ultra* (with a *Nvidia G80* processor) GPU (year 2007); see Table 2 for details. The GNU `gcc` 4.1.2 compiler is employed in the evaluation. The multithreading capabilities of Goto BLAS were enabled so that the two cores in the Intel CPU cooperate in solving the linear algebra operations.

Table 2. Description of the hardware used in our second experimental study

	CPU	GPU
Processor	*Intel Core 2 Duo*	*Nvidia GeForce 8800 Ultra*
Codename	Crusoe E6320	G80
Clock frequency	1.86 GHz	575 MHz
Peak performance	14.9 GFLOPS	520 GFLOPS
Memory speed	2×333 MHz	2×900 MHz
Bus width	64 bits	384 bits
Max. bandwidth	5.3 GB/s	86.4 GB/s
Memory	1024 MB DDR	768 MB DDR
Bus	PCI Express x16 (4 GB/s transfer rate)	
Year	2007	2007

The implementations of the linear algebra routines in the the CUBLAS library ([2]) were used in the evaluation. This is a library developed by Nvidia, implemented on top of CUDA, and optimized for unified graphics architectures as the *Nvidia G80*. The experimental evaluation showed that the implementations in CUBLAS outperformed our implementations using Cg.

For the convolution filter, we implemented a tuned version using CUDA, with intensive use of the fast shared memory per group of SP in order to optimize performance. We also applied other optimization guidelines proposed in [3], and common in the CUDA programming paradigm. On the CPU side, an optimized, BLAS-based implementation of the bidimensional convolution filter was used. This type of implementation is fully optimized with respect to the memory and SSE unit, so the comparison is considered to be fair.

5.3 Experimental Results

Input Data reutilization: `SGEMM` **vs.** `SGEMV`

Figure 6 (left-hand side) shows the performance of routine `SGEMM` on both platforms. Although this is not the most appropriate algorithm for the GPU (indeed, it only delivers about 20% of the peak power of the GPU), the performance on that platform is roughly 10 times higher than that obtained on the CPU.

The impact of the data transfer bottleneck in the performance of a routine is higher when its computational load decreases. For example, Figure 6 (right-hand side) illustrates the performance of routine `SGEMV`. The decrease in the GFLOPS rate is higher in this case when the transmission time is included. This difference is so important for this routine that, in case the transfer time is considered in the evaluation, the performance is lower on the GPU than on the CPU. When transfer times are not considered, the implementation on the CPU outperforms the CUBLAS implementation for large stream dimensions.

Note the different behavior of the `SGEMV` routine executed on CPU and on GPU. While on a general purpose processor the peak performance is achieved for relatively small amounts of data, obtaining worse results for bigger vectors, the maximum performance on GPU is always attained for large vectors. In fact, the optimized implementations of the BLAS, such as GotoBLAS, exploit very well the sophisticated cache systems of the most modern processors, and thus

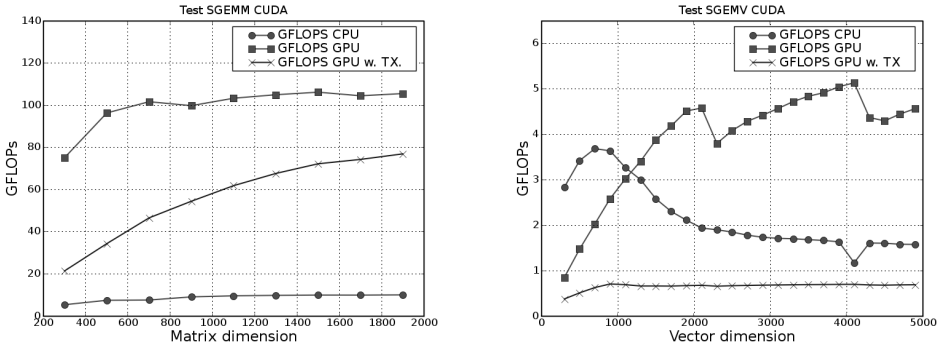


Fig. 6. Performance of routines `SGEMM` (left-hand side) and `SGEMV` (right-hand side) on the *Intel Core 2 Duo* CPU and the *Nvidia G80* GPU

benefits the computation with small matrices. On the other hand, GPUs do not present such advanced cache memories; this fact, and the stream-oriented architecture of this class of processors, benefit the computation over big amounts or streams of data, attaining poor results for small vectors.

Comparing routines `SGEMM` and `SGEMV`, the introduction of a sophisticated memory hierarchy in the *Nvidia G80* diminishes the impact of the data reutilization. The results for routine `SGEMM` are better when we compare them with CPU implementation than the results we obtain for routine `SGEMV`. The introduction of cache memories is one of the main differences between both generations of GPU and, from the previous results, we can conclude it has an important influence in the performance of general-purpose algorithms on GPUs with unified architectures.

Computational Load Per Stream Element: BLAS-1 Routines

The amount of computational load per stream element is also critical in this class of architectures. Figure 7 reports the results for routines `SAXPY` and `SSCAL`. Compared with the results attained for the classical architectures in Figure 4, although being better in absolute terms, the behaviors are similar: despite being stream-oriented algorithms, without any type of input data reutilization, the results are not comparable with those obtained by the tuned implementations in GotoBLAS. As occurred in previous experiments, results are better for a more computationally intensive algorithm such as `SAXPY`, attaining better results than `SSCAL`. The transfer time is more relevant in this case, as the computational load of the algorithms is quite low compared with that on more computationally intense routines, such as `SGEMM`.

Bidimensional Convolution Filter

Figure 8 shows the results obtained for the application of a convolution filter on a 512×512 image and variable filter size. This application again presents

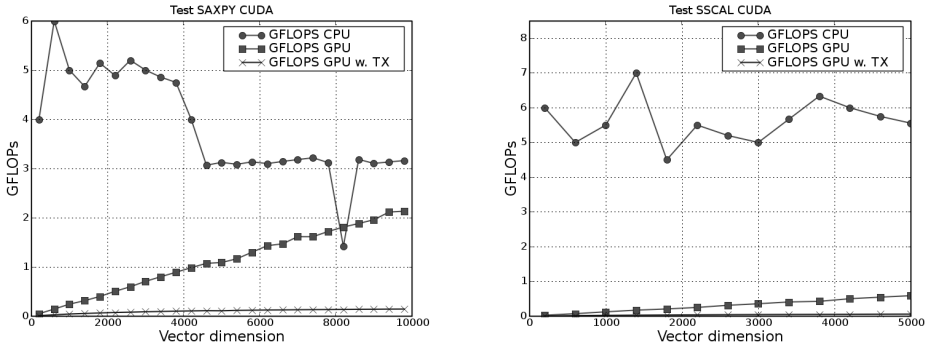


Fig. 7. Performance of routines `SAXPY` (left-hand side) and `SSCAL` (right-hand side) on the *Intel Core 2 Duo* CPU and the *Nvidia G80* GPU

the most favorable properties for its execution on current GPU architectures, attaining results up to 20 times better than those achieved for the same routines on a CPU. This is, in fact, the highest speedup achieved in our study.

Impact of Data Transfers

Data transfers were not a critical stage for the past generation GPUs. However, from the empirical results extracted for the most modern generation of graphics processors, we have proved that communication through the PCIExpress bus is now a factor to be considered.

The impact of the transfer time is larger for the unified architecture compared with non-unified architecture, with less powerful interconnection buses. In fact, the peak performance of the *Nvidia G80* is about 20 times higher than that of the *Nvidia NV44*, but the speed of the interconnection bus in the unified platform is only twice as fast as the one in the non-unified platform. This is a major bottleneck in current graphics platforms, and determines that GPU algorithms

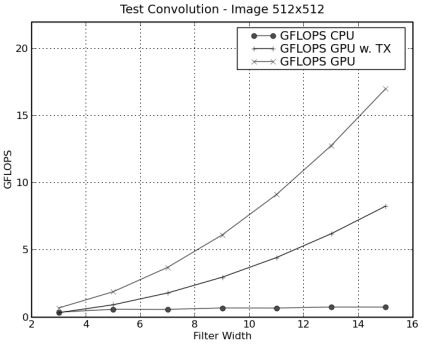


Fig. 8. Performance of the implementations of the convolution filter on the *Intel Core 2 Duo* CPU and the *Nvidia G80* GPU

must be redesigned to reduce the communications so that data in video memory is reused as much as possible before sending them back to RAM.

The latest GPU models from NVIDIA support the overlapping between memory transfer and computation on GPU, making it possible to hide the data transfer bottleneck for some operations. Unfortunately, the tested GPU did not support this feature.

6 Conclusions

We have presented a study of the properties which favor efficient execution of general-purpose algorithms on a graphics processor, considering both classical and unified architectures.

GPUs from previous generations, with classical architecture, are suitable for certain types of general-purpose algorithms with three basic characteristics: low input data reutilization, high data level parallelism, and high computational load per stream element. Despite their high computational power, the graphics-oriented nature of this class of hardware carries a set of limitations at the architecture level which ultimately limit the performance of certain types of algorithms like, e.g., routines from BLAS. On the other hand, GPUs of this nature obtain remarkable results for general-purpose algorithms which exhibit the three properties specified above, outperforming in this case the CPU.

The improvements introduced in the new generation of GPU (unified architecture, higher processing units replication, more sophisticated memory hierarchies, etc.) have increased the efficiency of this hardware to execute also for general-purpose algorithms. In fact, current GPUs deliver higher performance than that of timely CPUs in many applications.

Therefore, the last generation of GPUs appears as a high performance and low cost co-processing platform for a larger variety of applications. The emergence of general-purpose languages that facilitate their programming makes them even more interesting hardware from general-purpose computations. Nevertheless, GPUs still present some limitations in general-purpose computing such as numerical precision, data transfer stages, memory hierarchies not as sophisticated as CPU ones, etc. All this makes necessary to evaluate carefully the suitability of GPU as an accelerator for calculations.

Acknowledgments

This work has been supported by the CICYT project TIN2005-09037-C02-02 and FEDER. Francisco Igual-Peña is supported as well by a research fellowship from the *Universidad Jaume I of Castellón* (PREDOC/2006/02).

References

1. Barrachina, S., Castillo, M., Igual, F.D., Mayo, R., Quintana-Ortí, E.S.: Evaluation and tuning of the level 3 CUBLAS for graphics processors. In: Workshop on Multithreaded Architectures and Applications, MTAAP 2008 (2008)

2. NVIDIA Corp. NVIDIA CUBLAS Library (2007)
3. NVIDIA Corp. NVIDIA CUDA Compute Unified Device Architecture. Programming Guide (2007)
4. Fatahalian, K., Sugerman, J., Hanrahan, P.: Understanding the efficiency of GPU algorithms for matrix-matrix multiplication. Graphics Hardware (2004)
5. Basic Linear Algebra Subprograms Technical (BLAST) Forum. Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard (2001)
6. Galoppo, N., Govindaraju, N., Henson, M., Monocha, D.: LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware. In: ACM/IEEE SC 2005 Conference (2005)
7. Goto, K., Van de Geijn, R.: High-performance implementation of the level-3 BLAS. ACM Transactions on Mathematical Software
8. Govindaraju, N., Lloyd, B., Wang, W., Lin, M., Manocha, D.: Fast computation of database operations using graphics processors. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pp. 215–226 (June 2004)
9. Hong, J.Y., Wang, M.D.: High speed processing of biomedical images using programmable GPU. In: 2004 International Conference on Image Processing, ICIP 2004, 24-27 October 2004, vol. 4, pp. 2455–2458 (2004)
10. Larsen, E.S., McAllister, D.: Fast matrix multiplies using graphics hardware. In: Supercomputing, ACM/IEEE 2001 Conference, p. 43 (November 2001)
11. Moravánszky, A.: Dense matrix algebra on the GPU (2003)
12. Ruiz, A., Sertel, O., Ujaldon, M., Catalyurek, U., Saltz, J., Gurcan, M.: Pathological image analysis using the GPU: Stroma classification for neuroblastoma. In: Proceedings IEEE Intl. Conference on BioInformation and Bio Medicine (2007)

Optimised Computational Functional Imaging for Arteries

Ramiro Moreno[1], Ming Chau[3], Shirod Jeetoo[1], Franck Nicoud[2], Frédéric Viart[3], Anne Salvayre[1], and Hervé Rousseau[1]

[1] Institut de Médecine Moléculaire de Rangueil (I2MR), équipe 10
CHU de Rangueil, Toulouse, France
moreno.r@chu-toulouse.fr
[2] Institut de Mathématiques et Modélisations de Montpellier (I3M),
Université de Montpellier II, France
[3] Advanced Solution Accelerator (ASA), Montpellier, France

Abstract. The general framework of the Optimised Computational Functional Imaging for Arteries (OCFIA) program is to introduce high-performance scientific computing in the medical domain with the aim to rationalize therapeutic decisions in respect to vascular diseases yet poorly understood. More precisely, it consists in coupling medical imaging techniques, essentially morphological, with scientific computing, through Computational Fluid Dynamics (CFD), to yield functional imaging, thus providing to physicians a better quantitative knowledge of the biomechanical state (field of speeds, pressure, wall and Stent Graft loads ...) to the patients.

1 Introduction

Risk factors for cardiovascular disease (hypertension and high cholesterol) and their role have been identified, but cannot explain the observed localised occurrence and the progression of the disease (stenosis, aneurysm rupture, aortic dissection). Currently, available techniques such as Computed Tomography (CT), Magnetic Resonance Imaging (MRI) and Ultrasound (US) do not allow accurate determination of the complex velocity distribution and biomechanical load on the arterial wall. Nevertheless there is not doubt that medical imaging is an essential tool for the understanding of these pathological processes. Cardiovascular disease is clearly multi-factorial and it has been shown that deviations of the normal velocity field (changes in wall shear stress) play a key role [Caro,1969]. Despite many hemodynamic studies carried out with models of arterial bifurcations, especially the carotid artery bifurcation, the precise role played by wall shear stress (WSS) in the development and progression of atherosclerosis remains unclear. Still, it is certain that the mechanical load induced by the fluid on atherosclerotic plaques and their surrounding tissues is of the utmost importance for predicting future rupture (culprit plaques) and preventing ischemic events [Corti,2002]. In the same way, the risk of rupture of an aortic abdominal aneurysm (AAA) depends more on biomechanical factors than simply on the aneurysm diameter.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 420–429, 2008.

Although clinical decisions are based only on the latter today, wall tension is a significant predictive factor of pending rupture [Aaron, 2000].

Computational Fluid Dynamics (CFD) techniques can provide extremely detailed analysis of the flow field and wall stress (shear & tensile) to very high accuracy. New advances in simulation techniques could make a significant contribution to a better quantitative knowledge of the biomechanical condition of the arteries and lead to a new understanding via deepened insights into these conditions. Advanced simulations could potentially be used for predicting plaque and aneurysm rupture, improving endovascular prosthesis design, as well as for guiding treatment decisions by predicting the outcome of interventional gesture (i.e. stent-coil technique).

However, applying computational fluid dynamics (CFD) to actual pathological regions of the arterial tree is very challenging and has never been done so far with sufficient accuracy and time efficiency to be useful in the clinical practice. Today's medical research is strongly linked with advances in parallel and high performance computing. Ambitious research programs such as NeuroSpin (Saclay, France) are feasible thanks to the support of the computational ressources of CEA (9968, 63 TFlops, 7th in the top 500 supercomputers). Data storage, visualization and grid infrastructure are also key issues. The development of Virtual Vascular Surgery is encouraged by grid computing consortiums such as Crossgrid. There is no doubt that advances in computational ressources and infrastructure will benefit to the medical community. In return, more and more challenging applications will rise and stimulate research and development.

We present a complete, optimized calculation chain whose input come from an entirely non-invasive 4D MRI protocol that provides time varying geometry and flow rates and output is a realistic functional imaging description of the arterial tree region of interest. Preliminar results were obtained through parallel computing with AVBP code (CERFACS, Toulouse, France) and classical Arbitrary Langrangian Eulerian (ALE) formulation.

2 Materials and Methods

2.1 Image Acquisition: MRI Protocol

All the images were obtained with a 1.5 T MR scanner (Intera; Philips Medical Systems, the Netherlands) with a 5-element phased-array coil for signal reception (Sense Cardiac, Philips Medical Systems).

The objective of following processing steps was to translate the image data set into patient-specific conditions (boundary conditions) suitable for CFD calculations.

MR Angiography. The routine injected (Gd-DTPA, Magnevist, Schering, volume injected 20mL, injection rate 5mL/s) T1-Full Field Echo sequence was performed on sagittal-oblique planes, parallel to the major aortic axis, in order to cover the whole aorta geometry (field of view, 450x450x126mm) with a spatial resolution of 0,88x0,88x1.80mm^3. This acquisition was trigged to the patient ECG, 430ms after the R wave (diastolic phase).

Single Slice 2D MR Cine Imaging. A dynamic balanced Steady State Free precession (b-SSFP) sequence, was performed on transverse planes, to cover the thoracic aorta and segment the cardiac cycle in 20-40 phases.

MR flow quantification. 2D Phase-Contrast (PC) sequences, performed orthogonal to the vessel axis, provided the velocity inlet profiles at the ascending, descending aorta, and supraaortic vessels. Supplementary transversal-oblique and sagittal-oblique acquisitions, respectively ortogonal to the short and long aortas axis, were performed to compare quantitatively the velocity results from the CFD.

2.2 Image Data Processing

All of the next procedures described in this chapter were developed in-house in a Matlab language (The Matlabworks, Inc) with some C compiled routines (mexfiles) integrated to main sources. Some of them (affine coregistration and high dimensional warping are initially developped by John Ashburner into SPM (statistical parametric mapping toolbox University College London, 2000), an image processing toolbox dedicated to computational neuroanatomy. The images were initially obtained in DICOM format and were converted to the Analyze-7 format (Mayo Clinic, Rochester, USA., http://www.mayo.edu/bir/) for filtering and non-linear-transformation operations. The meshes were initially built in an ASCII format (*.cas) into the Amira 4.1 environement(TGS, Mercury Computer Systems, USA). The final set of moving meshes and hemodynamic boundary conditions are exported into the correct AVBP format (CERFACS, Toulouse, France, http://www.cerfacs.fr) in order to perform CFD runs. Finally, all post-processing steps before CFD step were operated with a Pentium(R) Duo 3,4 GHz processor with 1.5Go RAM. The geometrical post-processing operations are highlighted on the figure 1.

Filtering. MR image noise affects the post-processing algorithms. Filtering can be used to limit image noise but current filters reduce spatial resolution. In this work we applied a selective blurring filter to anatomical MR data. The filtered static data acquisition are submited to 3D Level Set algorithm (see section Initial Mesh) for a segmentation step. Compared to other classical filters, this filter achieves the best compromise between spatial resolution and noise reduction [Gensanne, 2005]. The homogeneous regions must have the same gray level, so if the noise (gradient) is less than 2xSNR (95%), the selective blurring filter apply a hard smooth (weighting=1). The treatment is different for the fine details where gradient is greather than 2xSNR the filter gradually weight the smooth according to the image neighbours gradient (weighting=1/gradient).

Initial Mesh. After this stage of filtering, the anatomical surface can be extracted by means of level-set/fast marching methods [Sethian, 1999] that accurately model the complex surfaces of pathological objects. With the static geometry acquisition, the level set methods offer a highly robust and accurate

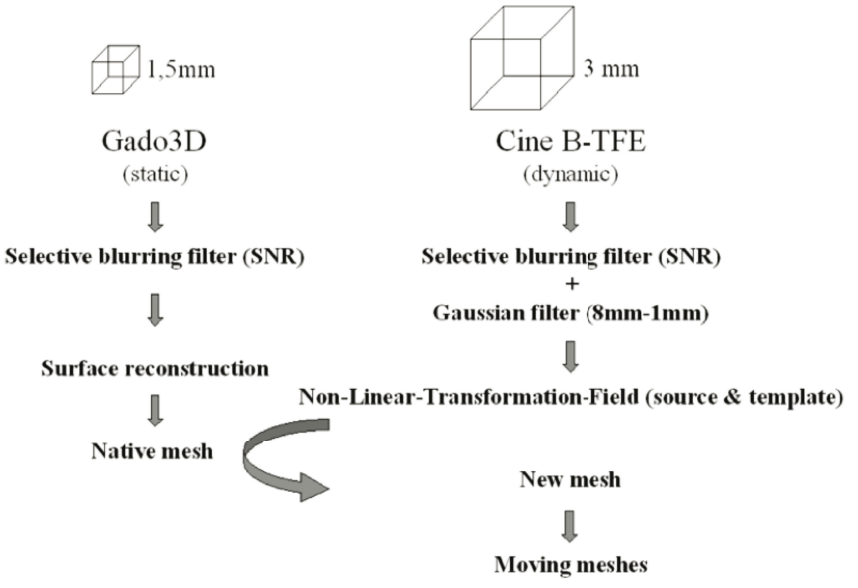


Fig. 1. Postprocessing steps for moving mesh preparation from MRI acquisitions

method for tracking the interface in a volume of interest coming from the static and contrast-enhanced MRI acquisition.

Given an initial position for an interface γ_0 (endovascular surface), where γ is a closed surface in R^3 , and a speed function F which gives the speed of γ in its normal direction, the level set method takes the perspective of viewing γ as a zero level set of a function ϕ from R^3 to R whose evolution equation is eq. 1 and where ϕ represents the distance function.

$$\begin{cases} \phi_t - F|\nabla\phi| = 0 \\ \phi(x, t = 0) = \pm d(x) \end{cases} \tag{1}$$

The filtered static data acquisitions (T1-Full Field Echo sequences) are submitted to the in-house 3D Level Set algorithm for a fine geometrical extraction step (Matlab 7.0, the MathWorks, Inc). An initial computational grid was obtained by the discretization of this geometry (Amira 4.1).

Moving Mesh. Wall movements were imposed to the initial grid according to cine scan acquisition (b-SSFP), by means of the non linear transformation field algorithm of SPM-2 toolbox (Matlab 7.0, the MathWorks, Inc). A tetraedral moving grid was built [Moreno, 2006] according to estimated non-linear deformation fields (Fig. 2 shows the mesh and its deformation on a section). Each phase of the transformation process consists in estimating the defomation between the native and a target image. Therefore, the whole transformation is completed when the deformations to all the target images of a cardiac cycle are computed. The meshes used at each time step of CFD simulation are obtained

by applying the computed deformation (according to cine scan images) to the native mesh (obtained from injected sequence).

Specifically, the optimization approach is used here. We are trying to find the best compromise between effective transformation, by a function F_2 (eq.3), and regular transformation, by a function F_1 (eq.2) which corresponds to the sum of local volume variations ($\frac{\Delta V}{V} = |J| - 1$). The deformation gradient tensor J is computed according to the coordinates of $T(x)$. Optimization process consists in finding T which minimizes a linear combination of F_1 and F_2 (eq.4).

$$F_1 = \int_\Omega (|J| - 1)\, d\Omega + \int_\Omega \left(|J|^{-1} - 1\right) d\Omega \tag{2}$$

$$F_2 = \int_\Omega [I_{source}(x) - I_{target}(T(x))]^2 d\Omega \tag{3}$$

$$F = \lambda F_1 + F_2 \tag{4}$$

The derivative of F is computed with symbolic calculation tool and a gradient algorithm is used. The phases of the transformation process (Fig. 2) can be computed independently. This is a strightforward parallelization.

Hemodynamic Boundary Conditions. A region of interest (ROI) defined the vascular area on the Phase Contrast (PC) quantitative images to extract velocity data samples. A curve fitting calculation was applied to this data set using the general Fourier model in order to obtain a time-dependent function. This process was applied to all the orthogonal sections to the short aorta axis (Fig. 3). The areas corresponding to the ascending, descending aorta and supraaortic vessels provided inlet and oulet conditions for CFD, while the mid descending aorta was used for the control. The arrow on the right side image points out a strongly turbulent flow. Note that these MRI images give the vertical component of velocity.

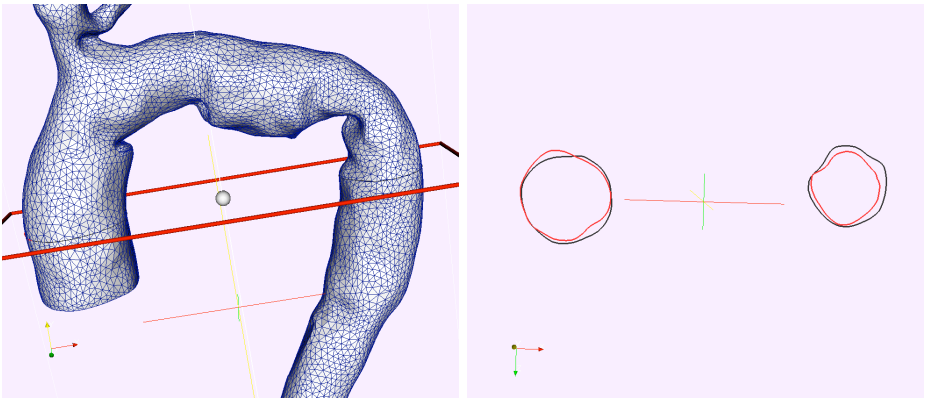


Fig. 2. Moving meshes (2 cardiac phases)

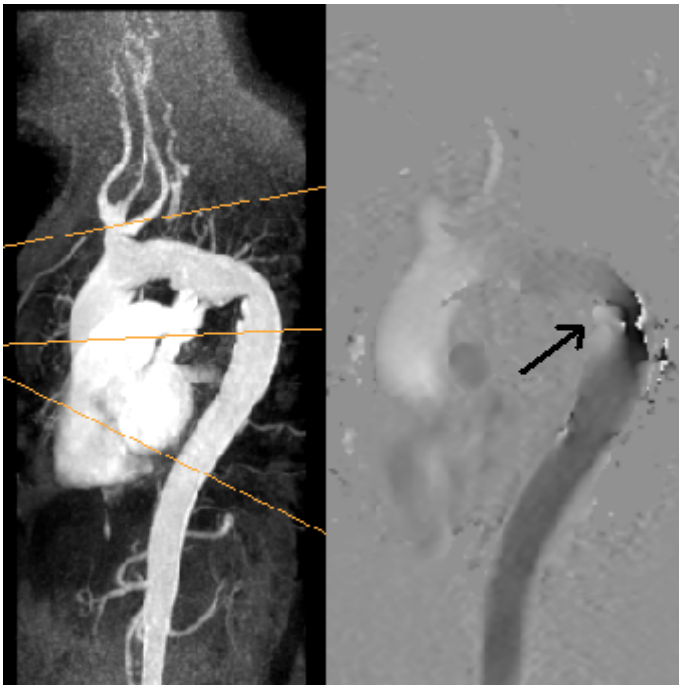


Fig. 3. Phase contrast aquisition for the inlet/oulet velocity evaluation

2.3 CFD Application

A first numerical model was developed [Nicoud, 2005] in order to assess the wall shear stress changes after endovascular stenting. In this approach, the fully coupled fluid–structure problem is replaced by a simpler fluid problem with moving boundaries. The NavierStokes equations were solved numerically with an appropriate finite-element-based method which handles time-dependent geometries. The main result supports the idea that stenting can induce endothelial dysfunction via haemodynamic perturbations. From this study, which enabled us to check the feasibility of an uncoupled CFD, we widened the problem with the more complex case of vascular geometries.

The flow simulations were performed using the finite volume (FV) method, as implemented in the AVBP code (CERFACS, European Center for Research and Advanced Training in Scientific Computation, Toulouse, France).

The FV method used in the code solves the full Navier-Stokes equation, who governs the flow, by an efficient explicit Arbitrary Lagrangian Eulerian (ALE) formulation (Fig. 4), which allows to impose the tetraedral moving grid within cardiac cycles.

Hemodynamic conditions (time-dependent functions) were synchronized with the wall motion and were imposed in the form of speed profiles at the entry (ascending aorta) and exit (descending aorta, supraortic vessels) of the numerical field (aorta district). Blood was assumed to be a homogeneous newtonian fluid

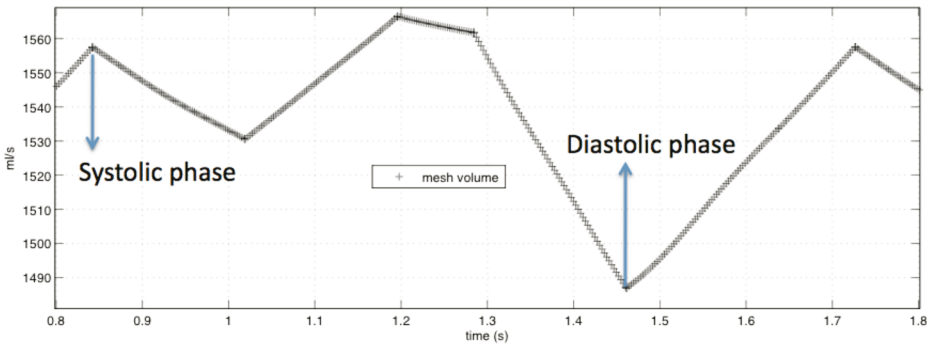


Fig. 4. Total mesh volume : run process (one ALE run per cardiac phase) for HPC

with a dynamic viscosity approximated as 4 cPoi and a density of 1050 kg/m3 (physiologic blood value in aorta and collaterals). The simulations began from an initially quiescent flow state and continued for a number of full cardiac cycles in order to allow the development of a fully periodic flow, representative of a regular heartbeat. It was found that the main features of the vascular flow field became stable within four cycles.

Uncoupled CFD results were performed in HPC system (IBM, Power4, Cines, Montpellier, France) and were controlled by the additional MRI quantitative-flow-imaging performed at intermediate levels during the examination. Geometric deformation field was validated by 3D visual correlation between moving mesh and cine scan imaging.

Tetraedral Grid Resolution (minimum length: 1mm; maximum lenght 2.3 mm), balanced between the grid independence and time efficiency conditions, was sufficient for preliminar results on Carotid Bifurcation and Thoracic Aorta CFD calculations.

3 Results

Thoracic aorta studies were performed in both volunteers and patient cases (Fig.5). Boundary conditions were flow controlled and Risk Factors were observed in relation with 'hot spots' in wall stress and hemodynamic results (shear stress and velocity). Patient data set was treated in 24 hours, according to the description of the table 1.

The data processing is virtually automatic, the only manual interventions are corrections of errors of the native geometry and preparation of boundary conditions during the extraction process. Normal WSS values calculated by CFD on healthy ascending regions on patients were similar to data available on literature [Efstathopoulos, 2008] performed on healthy subjects by phase-contrast MRI flow measurements and straightforward methodologies based on Poiseuille's theory of flow. Systolic (0.3 ± 0.2 N/m2 compared to 0.4 ± 0.2 N/m2) and diastolic (0.065 ± 0.04 N/m2 compared to 0.11 ± 0.07 N/m2). The same group of patients presented abnormal WSS values on landing zones (+12% at systole,

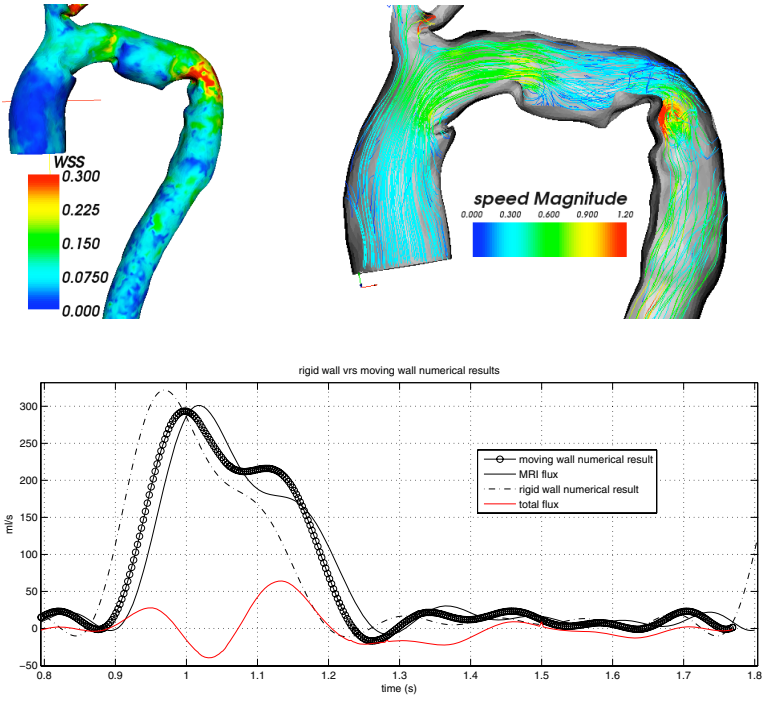


Fig. 5. Stented thoracic aorta : wall shear stress, stream lines velocity, control fow at intermediate level for rigid and moving wall

+35% at diastole), 'kinked' zones (+38% ± 6% at systole and +25 ± 11% at diastole) and collateral (neck vessels) zones (+6 ± 2% at systole).

On the 4D-CFD models, the patterns of the aortic velocity (streamlines, vector fields) were clearly observed. Localized sharp increases - defined as hot spots - were quantitatively depicted and referred to on the color-coded scale (cm/sec) at the side of each model. These velocity hot spots were found at the zones of sharp diameter transitions (210±30 cm/sec at systole and 50±30 cm/sec at diastole) as observed in the stenotic fluidodynamic patterns. This was observed both in the native aorta as well as in the stent-graft. The vector fields analysis in m/s well-represented the velocity patterns at specific targeted levels and could be reformatted according to the users purpose.

Characterizing the dynamic components of blood flow and cardiovascular function can provide insights into normal and pathological physiology. Both vascular anatomy and hemodynamics in arterial pathologies are of high interest for the understanding of the development and progression of diseases as well as justifying the therapeutic decision. During the last decade, many experiences have been focusing on the relevance of the flow dynamics in the prediction of abdominal aneurysm growth and rupture[Ekaterinaris, 2006]. The aortic wall stresses have been proposed as new, more precise markers for patients monitoring

and predicting rupture, and, very recently, novel non-invasive applications have shown up to estimate the parietal tension

4 Discussion

Following the extraction from the Level Set method, we found that it was restrictive to make weary calculations to extract the initial geometry. Sethian's method is indeed very interesting but it depends heavily on the image quality. Partial volume artefacts increase with the thickness of the slice for a given imaging volume and results in blurring in regions where several vessels cross. An extraction method with the Level Set technique is erroneous because it sees a single vessel rather than distinguishing them. We try to solve this problem by asking experienced radiologists to make the geometry extraction through a threshold setting procedure. It revealed to be very efficient, the extraction being made instanteneously with the possibility for error correction by the user.

The flow curves obtained in the reference plan reveals the importance of taking into account the mobile wall in a realistic case. The result obtained in a rigid mesh is far better compared to the MRI measures. Yet the results obtained on a moving mesh are very similar to the control measures. It seems obvious that a realistic calculation needs to be done according to the rheology of the wall.

The calculations of deformation fields require a single pair of reference volumes (source and template). The computations have been performed on a dual-core processor (table. 1). The N deformations to be computed are dispatched on the processors. As they are independant, no communication is required during the parallel computing. It is thus trivial to reduce the time to $1/n$ if we have such processors at our disposal.

Table 1. Time consuming for the general clinical case

Chain element process	Time consuming
MRI protocol.	30 min
Level Set geometry extraction (not parallel).	60 min
Native mesh prepartation and correction (interactive, manual).	120 min
Moving mesh estimation process (not parallel).	600 min
Moving mesh estimation process (parallel@2proc).	300 min
CFD (HPC, 24 proc).	600 min
TOTAL	1410 min (23h30 min)

Table 2. Run characteristics

parameter	value
number of tetraedral elements	145848
number of iterations for convergence	2.6182e+08
fixed time step for moving wall	0.55E-04 s
time step for rigid wall	0.32E-04 s
delta Volume for moving mesh	84 mL

5 Conclusion

The proposed approach permits the computation of the blood flow under realistic in vivo, time evolving and flow controlled conditions. It is much simpler than the full coupled fluid-structure problem and has the potential to provide a better picture of the specific hemodynamic status. The method gives a direct way to impose realistic wall interaction to hemodynamic time boundary conditions, from medical examinations performed into a simple MR protocol. A future improvement in the processing chain described in this work will facilitate biomechanical functional imaging in less than 8 hours, which would be useful for clinical practice. Insights about the physiopathology of some arterial diseases and endovascular treatment are also expected.

References

[Caro,1969] Caro, C., Fitz-Gerald, J., Schroter, R.: Arterial wall shear and distribution of early atheroma in man. Nature 211 223, 1159–1160 (1969)

[Corti,2002] Corti, R., Baldimon, L., Fuster, V., Badimon, J.J.: Endothelium, flow, and atherothrombosis, Assessing and Modifying the vulnerable atherosclerotic plaque, American Heart Association, ed Valentin Fuster (2002)

[Aaron, 2000] Hall, A.J., Busse, E.F.G., McCarville, D.J., Burgess, J.J.: Aortic wall tension as a predictive factor for abdominal aortic aneurysm rupture: improving the selection of patients for abdominal aneurysm repair. An Vasc. Surg. 142, 152–157 (2000)

[Gensanne, 2005] Gensanne, D., Josse, G., Vincensini, D.: A post-processing method for multiexponential spin spin relaxation analysis of MRI signals. Physics in Medicine and Biology 50, 1–18 (2005)

[Sethian, 1999] Sethian, J.A.: Level Set Methods and Fast Marching Methods. Cambridge University Press, Cambridge (1999)

[Nicoud, 2005] Nicoud, F., Vernhet, H., Dauzat, M.: A numerical assessment of wall shear stress changes after endovascular stenting. Journal of Biomechanics 38(10), 2019–2027 (1999)

[Efstathopoulos, 2008] Efstathopoulos, E.P., Patatoukas, G., Pantos, I., Benekos, O., Katritsis, D., Kelekis, N.L.: Measurement of systolic and diastolic arterial wall shear stress in the ascending aorta. Physica Medica (2008)

[Moreno, 2006] Moreno, R., Nicoud, F., Rousseau, H.: Non-linear-transformation-field to build moving meshes for patient specific blood flow simulations. In: European Conference on Computational Fluid Dynamics, ECCOMAS CFD 2006 (2006), `http://proceedings.fyper.com/eccomascfd2006/documents/59.pdf`

[Ekaterinaris, 2006] Ekaterinaris, J.A., Ioannou, C.V., Katsamouris, A.N.: Flow dynamics in expansions characterizing abdominal aorta aneurysms. Ann. Vasc. Surg. 20, 351–359 (2006)

Memory Locality Exploitation Strategies for FFT on the CUDA Architecture

Eladio Gutierrez, Sergio Romero, Maria A. Trenas, and Emilio L. Zapata

Department of Computer Architecture
University of Malaga
29071 Malaga, Spain
{eladio,sromero,maria,ezapata}@ac.uma.es

Abstract. Modern graphics processing units (GPU) are becoming more and more suitable for general purpose computing due to its growing computational power. These commodity processors follow, in general, a parallel SIMD execution model whose efficiency is subject to a right exploitation of the explicit memory hierarchy, among other factors. In this paper we analyze the implementation of the Fast Fourier Transform using the programming model of the Compute Unified Device Architecture (CUDA) recently released by NVIDIA for its new graphics platforms. Within this model we propose an FFT implementation that takes into account memory reference locality issues that are crucial in order to achieve a high execution performance. This proposal has been experimentally tested and compared with other well known approaches such as the manufacturer's FFT library.

Keywords: Graphics Processing Unit (GPU), Compute Unified Device Architecture (CUDA), Fast Fourier Transform, memory reference locality.

1 Introduction

The Fast Fourier Transform (FFT) nowadays constitutes a keystone for many algorithms and applications in the context of signal processing. Basically, the FFT follows a *divide and conquer* strategy in order to reduce the computational complexity of the discrete Fourier transform (DFT), which provides a discrete frequency-domain representation $X[k]$ from a discrete time-domain signal $x[n]$. For a 1-dimensional signal of N samples, DFT is defined by the following pair of transformations (forward and inverse):

$$X = DFT(x)\colon\ X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \quad 0 \le k < N$$

$$x = IDFT(X)\colon\ x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-kn}, 0 \le n < N$$

where the powers of $W_N = e^{-j\frac{2\pi}{N}}$ are the so-called twiddle factors.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 430–443, 2008.

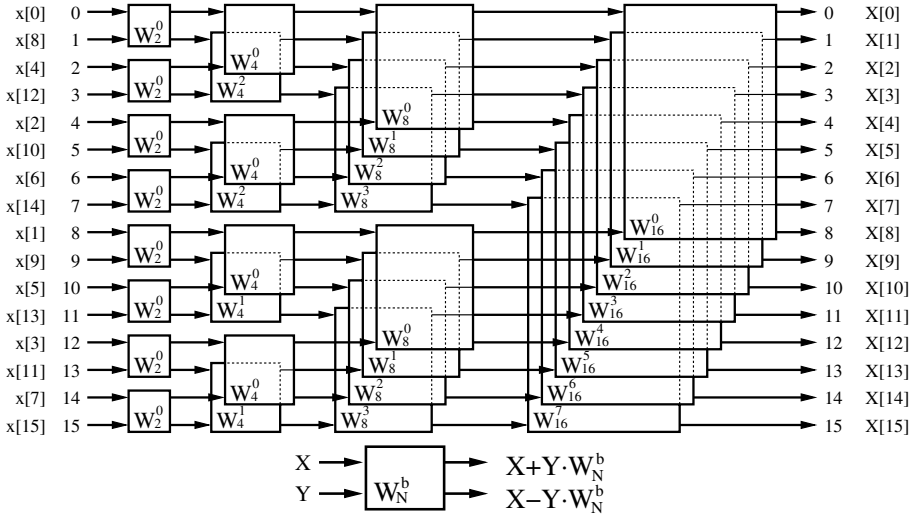


Fig. 1. Radix-2 decimation-in time FFT in terms of butterfly operators

The FFT organizes the DFT computations, as shown in Fig. 1, in terms of basic blocks, known as butterflies. The computation is carried out along $\log_2 N$ stages being computed N coefficients per stage. This way, the computational complexity is reduced to $\mathcal{O}(N \log_2(N))$ instead of $\mathcal{O}(N \times N)$ as inferred directly from the DFT definition.

Several configurational issues have been preset in Fig. 1. This configuration is known as *radix two* because butterflies operate on two inputs generating two transformed coefficients. Before the first stage, input coefficients are permuted in bit reversal order with the purpose of obtaining the right output arrangement. Such a rearrangement in time domain gives rise to the denomination *decimation-in-time* algorithm. This configuration is used the rest of the paper.

From the viewpoint of memory reference locality, we can observe that if the input coefficients are located into consecutive memory positions, the reference patterns of higher stages will exhibit poorer locality features than the lower ones. In addition, we must remark that if the input coefficients are permuted properly, it is possible to carry out one of the stages using the access pattern of another, simply by using the corresponding twiddle factors. Such an equivalence is depicted in Fig. 2 showing how 5^{th} and 6^{th} stages can be performed with the access pattern of the 3^{rd} and 4^{th} ones, after permuting the coefficients.

For subsequent use, we will denote $L_{(N,j,i)}(x)$ as the computation of j-th stage of a N-sample signal, but using the access pattern of the i-th stage (excluding the permutation) and $L_{(N,i)}(x) = L_{(N,i,i)}(x)$ the computation of the i-th stage with the proper pattern and twiddle factors. This way we can write the full FFT computation as $X = FFT(x) = L_{(N,s-1)}(...(L_{(N,1)}(L_{(N,0)}(P(x))))...)$, assuming that the number of samples is $N = 2^s$, and P represents the bit reversal permutation of the signal.

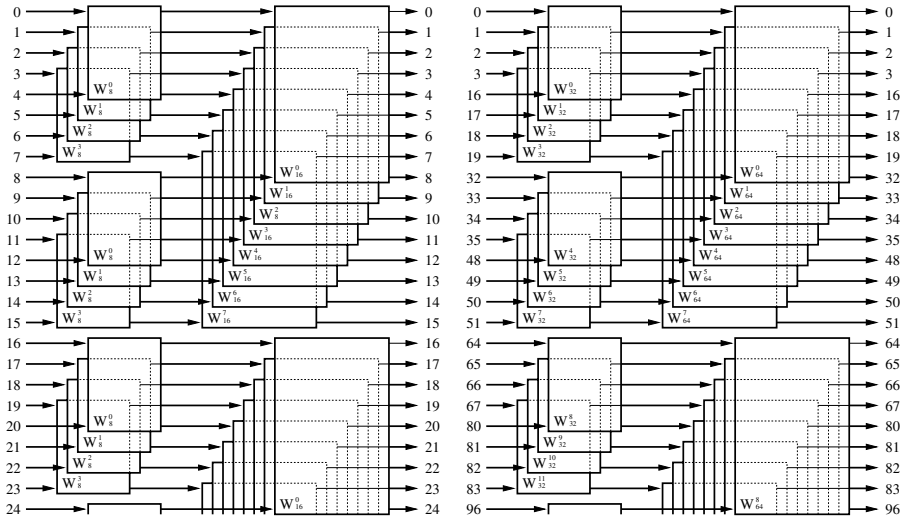


Fig. 2. Computing 3^{rd} and 4^{th} stages of the FFT (a); computing 5^{th} and 6^{th} stages of the FFT using the pattern of 3^{rd} and 4^{th} stages over a properly permuted input (b)

2 CUDA Programming Model

The Compute Unified Device Architecture (CUDA^{TM}) from NVIDIA®, is both a hardware and software architecture for issuing and managing computations on the GPU, making it to operate as a truly generic data-parallel computing device. An extension to the C programming language is provided in order to develop source codes.

From the hardware viewpoint, the GPU device consists of a set of SIMD (Single Instruction Multiple Data) multiprocessors each one containing several processing elements (processors), as shown in Fig. 3. Different memory spaces are available. The global device memory is a unique space accessible by all multiprocessors, acting as the main device memory with a large capacity. Besides, each multiprocessor owns a private on-chip memory, called shared memory or parallel data cache, of a smaller size and lower access latency than the global memory. A shared memory can be only accessed by the multiprocessor that owns it. In addition, there are other addressing spaces, omitted in the figure, for specific purposes: texture and constant memories.

CUDA execution model is based on a hierarchy of abstraction layers: grids, blocks, warps and threads (Fig. 4). The thread is the basic execution unit that is actually mapped onto one processor. A block is a batch of threads cooperating together on one multiprocessor and therefore all threads in a block share the parallel data cache. A grid is composed by several blocks, and because there can be more blocks than multiprocessors, different blocks of a grid are scheduled among the set of multiprocessors. In turn, a warp is a group of threads executing in an

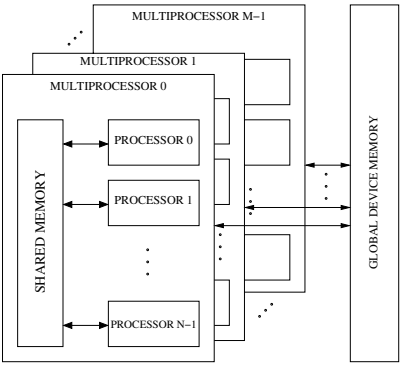


Fig. 3. Organization of processors and memory spaces in CUDA

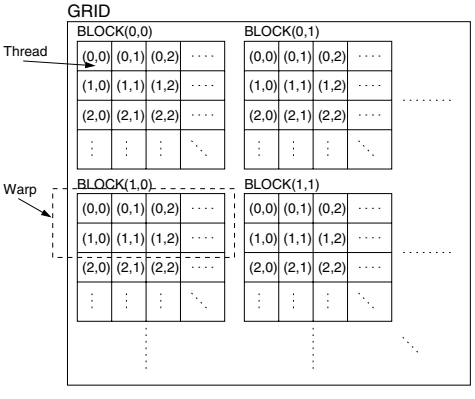


Fig. 4. Thread-based execution model in CUDA

SIMD way, so threads of a same block are scheduled in a given multiprocessor warp by warp.

Two kinds of codes are considered in the CUDA programming model: those executed by the CPU (host side) and those executed by the GPU, called kernel codes. The CPU is responsible of transferring data between host and device memories as well as invoking the kernel code, setting the grid and block dimensions. Such kernels are intended to be executed in an SIMD fashion over the processors.

Memory accesses and synchronization scheme are the most important aspects to take into account. Warp addresses issued by SIMD memory access instructions may be grouped thus obtaining a high memory bandwidth. This is known as coalescing condition. Otherwise, access will be serialized and the resulting latency will be difficult to hide with the execution of other warps of the same block. Global synchronization is not provided at the device side, only threads in a block can be waiting one to each other. Thus block synchronization mechanism must be explicitly implemented by the host through consecutive kernel invocations.

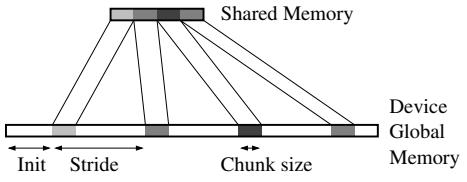


Fig. 5. Data transfer pattern between device and shared memory of `copy_in/copy_out` operations

3 Implementation Strategies for the FFT

In this section we analyze an FFT implementation using the programming model previously described. The goal is to obtain a high degree of parallelism taking into account system constrains, specifically those related to the memory hierarchy. The basic idea consists of mapping coefficients placed in global (device) memory into the data parallel cache (shared memory), performing all possible computations with these local data and then copying the updated coefficients back to the global memory. This process may be repeated with different mapping functions until all stages are done.

In order to be more precise we firstly introduce some useful functions describing the FFT implementations under study. These functions represent data transfers and transformations accomplished in a single shared memory.

Function `copy_in(ii,nc,sz,st)` copies a subset of signal coefficients from the device memory into consecutive positions of the shared memory, adding a padding when necessary to avoid memory bank conflicts. Its behaviour is depicted in Fig. 5. It starts from the `ii`-th coefficient and copies `nc` chunks of size `sz` coefficients separated by a stride `st`. Symmetrically, `copy_out(ii,nc,sz,st)` copies coefficients from shared memory back to the device memory. During the SIMD execution of these functions, each thread is in charge of transferring only a pair of coefficients. Observe that threads in a warp must access consecutive memory locations with the purpose of coalescing global memory accesses. Thus, the arguments of such functions describes how coefficients are accessed and hence if this transference fulfills coalescing criteria. In general, the chunk size must be a multiple of the warp size for an optimal transfer. Accesses are serialized when the chunk size is smaller.

The function `fft_level(i,j)` corresponds to the application of the operator $L_{(N,i,j)}(x)$ as described in section 1. Such a function is intended to be applied to the coefficient vector x, previously transferred to shared memory, and it operates in-place. The number of blocks of threads is $\frac{N}{2^r}$, where 2^r is the number of coefficients copied into shared memory. Executing this function in an SIMD way on a butterfly-per-thread basis, the b-th thread computes the b-th butterfly transformation, so a block performs 2^{r-1} butterflies, one per thread. Twiddle factors for this case are determined by the b-th butterfly of the j-th FFT stage, whereas the coefficients to be transformed are those involved in the b-th butterfly of the i-th FFT stage.

GPU side, one single block

```
copy_in(0,1,N,1);
for (i=0; i<s; i++) {
   syncthreads();
   fft_level(i,i);
}
copy_out(0,1,N,1);
```

Fig. 6. FFT of a signal fitting the shared memory

Let us analyze the naive case when the whole input signal fits into the shared memory of one SIMD multiprocessor ($N \leq 2^r$), whose implementation is shown in Fig. 6. After a bit reversal permutation, coefficients are transferred to the shared memory, then s invocations of `fft_level` are executed and finally coefficients are returned to the device memory. Note that as all the threads belong to the only block, global synchronization can be performed among threads. Besides, the original FFT scheme is applied locally (both `fft_level` arguments are equal).

The generic case of a signal whose size exceeds the available shared memory of a multiprocessor ($N > 2^r$) is discussed in next subsections. In this case several multiprocessors are involved, meaning that different blocks of threads must collaborate to perform the FFT.

3.1 Straightforward Approach

The first approach to be analyzed is a straightforward solution, but it exploits barely the locality features of the memory access pattern. As threads of different blocks cannot be synchronized within kernel code, required synchronizations must be carried out in the host side through successive kernel function invocations. Each kernel code invokes `copy_in` and `copy_out`. Between these two invocations, several `fft_level` stages need to be performed. The larger the signal size, the larger the number of butterflies operators and also the larger the number of FFT stages. Due to the fixed size of shared memories, the input signal must be distributed among the blocks of threads. Thus, a number of blocks of threads equal to $\frac{N}{2^r}$ will work with their corresponding set of disjoint coefficients. Let us consider each block with a set of 2^r consecutive complex coefficients. This way the first r FFT stages can be performed independently of the work of other blocks. Nevertheless, threads within the block must be synchronized before every stage in order to ensure that its input coefficients are updated by the previous stage. The remainder FFT stages involve coefficients located at a distance larger than 2^r, that is, their copies are located on different shared memories, on different multiprocessors. In order to proceed forward, coefficients must be properly rearranged. This fact involves a `copy_out` and a host synchronization prior to continue with the next stages.

At this point, all output coefficients of the r-th FFT stage can be found in the device memory. For the sake of a simplified discussion, let be $2^r \geq N/2^r$ ($2^s \leq 2^{2r}$), that is, the total number of FFT stages s is at most $2r$. As these

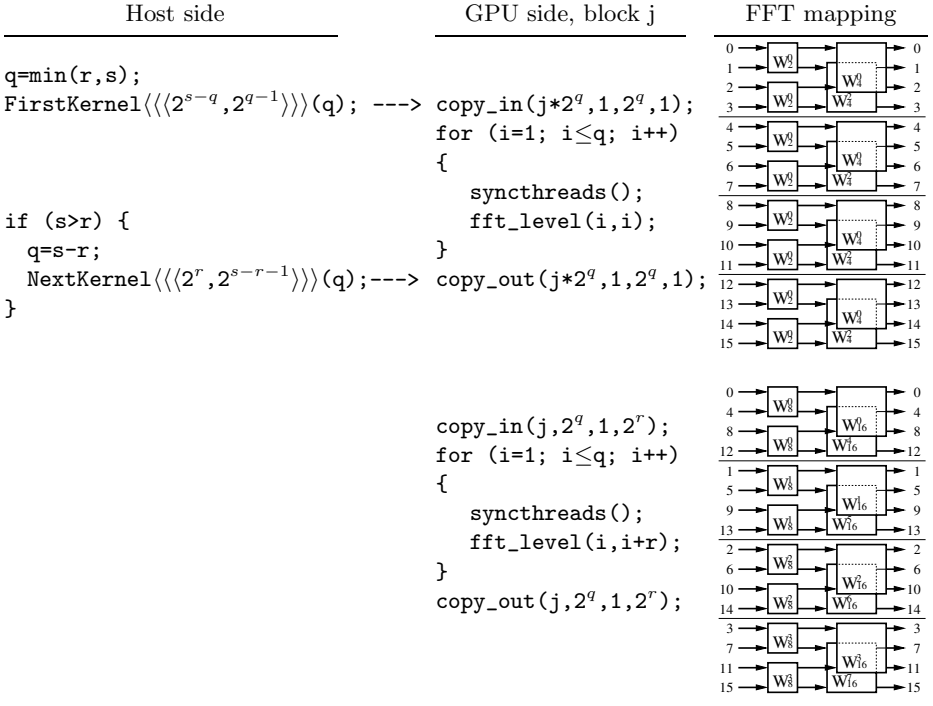


Fig. 7. Straightforward FFT implementation for large signals

r levels have been just computed, only r subsequent stages remain at most, and therefore only a new kernel invocation is needed. In general, $\lceil \frac{s}{r} \rceil$ kernel invocations will be required. Assigning the 2^r sequences of size 2^{s-r} coefficients with stride 2^r to different blocks (one sequence per block), all the $s-r$ remaining stages can be performed as shown in Fig.7. This pictorial example shows how a 16-samples FFT ($s = 4$) is performed for $r = 2$, that is 4 coefficients per block. Following the notation introduced by CUDA for its extended C language, the numbers enclosed in the triple angle notation ($\langle\langle\langle$`nB,nTpB`$\rangle\rangle\rangle$) stand for the total number of blocks and the number of threads per block respectively.

Observe that an important fact affects adversely the performance of the second kernel call (`NextKernel`). As threads are scheduled in warps behaving like gangs of threads that execute the same SIMD instruction, the memory addressing mode must follow a specific pattern for an efficient execution. In the case of global memory, threads of a same warp must access to consecutive memory locations, otherwise accesses are serialized. This condition is called coalescing requirement. The approach of Fig. 7 suffers from this lack of coalescing because memory locations accessed by copy operations do not contain chunks of consecutive coefficients. Observe that the third argument (size of chunks) of the copy functions in `NextKernel` invocation is set to one. This way, the first block in the example operates with vector (0, 4, 8, 12).

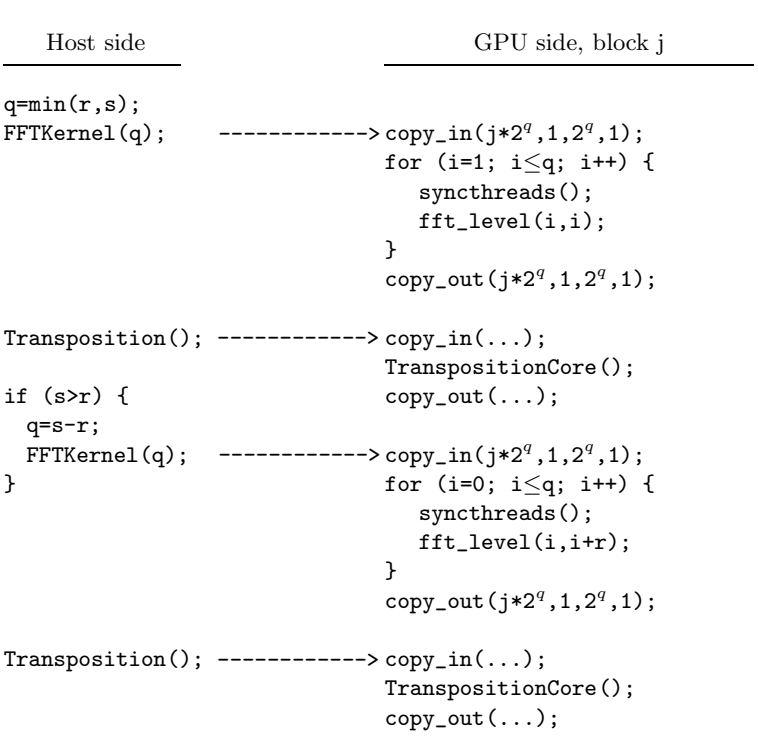


Fig. 8. FFT implementation for large signals using matrix transpositions

3.2 Transposition-Based FFT

A well known solution to this problem is to store the input signal in a 2D matrix ($2^{s1} \times 2^{s2}$ with $s = s1+s2$), 1D FFT is applied to every row (first $s1$ stages), then the matrix is transposed and finally 1D FFT is again applied to every row (last $s2$ stages). In order to apply correctly these last stages, a transformation of the transposed matrix is required as described in [6]. This step can be avoided if these 1D FFT stages use the corresponding twiddle factors of the original FFT higher stages as shown in Fig. 8. Note that input coefficients for the second invocation to the `FFTKernel` are now located on consecutive positions satisfying memory access coalescing demands, but this technique requires extra `copy_in/copy_out` operations for each transposition stage.

Broadly, a matrix transposition can be carried out in a block fashion by decomposing it into submatrices of size $m \times n$ fitting the shared memory. Submatrix (i, j) can be copied-in fulfilling the coalescing requirements because the m elements in the same row are consecutive. Once in the shared memory, the submatrix is transposed. Finally, the transposed submatrix is efficiently copied-out in its symmetrical position (j, i) as there are m chunks of n consecutive elements. Source codes for an efficient implementation of the matrix transposition can be found in the manufacturer's website [8].

For signal size larger than $2^r \times 2^r$ this approach uses a higher dimensional matrix representation of the coefficients. In general, for $2^{n \times r}$ coefficients a $n-$dimensional matrix is required. For example, if $2^{2r} < N \leq 2^{3r}$, signal coefficients are arranged in a 3D matrix. In this manner, 1D FFT is needed for each dimension, being necessary to do and undo transpositions not only for the second dimension but also for the third one.

3.3 Locality Improved FFT

With the purpose of improving the data locality in the higher levels of the FFT of large signals, we propose the technique described as follows. The key idea consists of transferring chunks of consecutive coefficients with a given stride among them, allowing the application of higher FFT stages using lower FFT stage access patterns. This technique is depicted in Fig. 9, where the left column corresponds with the host side code for a generic signal size, which has been unrolled to the particular case of two iterations matching a signal up to 2^{2r} samples. Observe that invocations to `NextKernel` are not preceded by any transposition and, what is more important, `copy_in/copy_out` operations meet the coalescing condition. This way, on avoiding transposition stages, the number of memory transfer operations is significatively reduced. The number of higher FFT stages that can be mapped on lower ones depends on the number of chunks (nC), in particular $log_2(nC)$ stages. Moreover, the number of chunks depends on the size of the chunks, which is determined by the number of threads of a warp (coalescing condition). For this reason, in the example of Fig. 9 the host invokes `NextKernel` two times, one half of the higher stages are performed in each invocation. Observe that the third argument (size of chunks) of the copy functions in `NextKernel` invocation is set to 2^{r-q} where q is the number of FFT stages to be computed. Therefore the lower the number of stages the higher the number of kernel invocations and so less reusability of data in the shared memory. Nevertheles if q is less than the warp size the coalescing gets worse.

By way of illustration, let us consider the case of an FFT of an input signal whose size is 256 coefficients (8 FFT radix-2 stages), running on a GPU with 8 threads per block assembled in 4 threads per warp and a shared memory with room for 16 coefficients per block. With this configuration, the whole FFT can be decomposed into 16 block of 16 consecutive coefficients (after a bit reversal permutation) performing the four first FFT stages. Then, coefficients must be rearranged in order to proceed with the next stages. As warps are made of 4 threads, the chunk size is fixed to 4 consecutive coefficients, but pairs of coefficients separated 2^4 are required, so the stride is 16. Function `copy_in(4j,4,4,16)` collects all coefficients for the j-th block, enabling it to perform 5^{th} and 6^{th} stages using the access patterns of 3^{rd} and 4^{th} stages by means of `FFT_LEVEL(3,5)` and `FFT_LEVEL(4,6)`. This is the same example shown in Fig. 2. Note that 5^{th} and 6^{th} stages can not be remapped onto stages 1^{st} and 2^{nd} because of their shared memory access pattern. This involves that stages 7^{th} and 8^{th} must be performed after a new rearrangement of the coefficients (`copy_in(4j,4,4,64); FFT_LEVEL(3,7); FFT_LEVEL(4,8)`).

Host side	Host side (unrolled)	GPU side, block j
`q=min(r,s);`	`q=min(r,s);`	
`FirstKernel(q);`	`FirstKernel(q);` --->	`copy_in(j*2^q,1,2^q,1);`
		`for (i=1; i≤q; i++) {`
		`syncthreads();`
		`fft_level(i,i);`
`p=0;`	`p=0;`	`}`
	`/*1st iteration*/`	`copy_out(j*2^q,1,2^q,1);`
`while (q<s){`	`if (s>r) {`	
`p=p+q;`	`p=p+q;`	
`q=min(r/2,s-p);`	`q=min(r/2,s-p);`	
`NextKernel(p,q);`	`NextKernel(p,q);` --->	`copy_in((j+j/2^q)*2^q,2^q,2^{r-q},2^p);`
`}`	`}`	`for (i=1; i≤q; i++) {`
		`syncthreads();`
		`fft_level(r-q+i,p+i);`
		`}`
	`/*2nd iteration*/`	`copy_out((j+j/2^q)*2^q,2^q,2^{r-q},2^p);`
	`if (s>r+r/2) {`	
	`p=p+q;`	
	`q=s-p-r/2;`	
	`NextKernel(p,q);` --->	`copy_in((j+j/2^q)*2^q,2^q,2^{r-q},2^p);`
	`}`	`for (i=1; i≤q; i++) {`
		`syncthreads();`
		`fft_level(r-q+i,p+i);`
		`}`
		`copy_out((j+j/2^q)*2^q,2^q,2^{r-q},2^p);`

Fig. 9. Improved-locality FFT implementation for large signals

According to the hardware specifications of the target platform, the maximum number of threads per block is 512, the maximum number of threads per warp is 32 and the shared memory size is 8 Kbytes, so 1024 complex coefficients fit. First 10 FFT stages ($r = 10$) are performed in the invocation of `FirstKernel`. In order to maximize the coalescing the chunk size should be a multiple of the maximum number of threads per warp. Since there are 32 chunks of 32 coefficients in 1024 coefficients, the 6^{th} stage is the first one onto which a higher stage can be mapped. This fact involves that only 5 higher stages can be done in each invocation to `NextKernel`. A lower number of threads per warp allows `NextKernel` to perform more stages, however the degree of fine-grained parallelism will decrease. In Fig. 9 the invocation to `FirstKernel` performs r stages whilst successive invocations to `NextKernel` perform at most $\frac{r}{2}$ stages. Although this technique can double the number of `NextKernel` invocations compared with the straightforward solution, that is, host side synchronizations, the improvement of coalesced global memory accesses is worthwhile because non-coalesced accesses are serialized (up to 32, the number of thread per warp).

4 Experimental Results

The locality-improved strategy for the 1D radix-2 complex FFT above discussed has been implemented and tested. Experiments have been conducted on a NVIDIA GeForce®8800GTX GPU, which includes 16 multiprocessors of eight processors, working at 1.35GHz with a device memory of 768MB. Each multiprocessor has a 8KB parallel data cache (shared memory). The latency for global memory is about 200 clock cycles, whereas the latency for the shared memory is only one cycle.

Codes have been written in C using the version 1.1 of CUDA™, released by NVIDIA®[8]. The manufacturer provides a platform–tuned FFT library, CUFFT, which allows the users to easily run FFT transformations on the graphic platform. The CUFFT library offers an API modelled after FFTW [2,3], for different kinds of transformations and dimensions. We have chosen CUFFT to be used with the purpose of measurement comparisons. Since the manufacturer recommends the transposition strategy for signals exceeding the supported signal size limit, CUFFT for supported signal sizes can be considered an upper limit for the transposition technique.

We have executed a forward FFT measuring the number of GigaFLOPS obtained. A common metric [2] considers that the number of floating point operations required by a radix-2 FFT is $5Nlog_2(N)$. Thus, if the number of seconds spent by the forward FFT is t_{FFT}, the number of GFLOPS for a N-sample signal will be $GFLOPS = \frac{5Nlog_2(N)}{t_{\mathrm{FFT}}}10^{-9}$. According to the CUFFT/FFTW interface, two dimensionality parameters are taken into consideration: the signal size (N) and the number of signals (b) of the given size to be processed. In literature that is known as a batch of b signals. For example, the transformation of four

Table 1. Measured GFLOPS for the CUFFT library and the proposed locality-improved FFT version (liFFT). N represents the number of coefficients of the transform for a batch of one single signal ($b = 1$). Void entries correspond to unsupported configurations due to memory constraints.

Single signal of N coefficients																
$\log_2(N)$	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
CUFFT	**0.43**	0.47	0.98	2.01	3.68	6.03	10.7	13.9	13.6	19.4	**23.0**	19.6	16.9	16.9	-	-
liFFT	0.35	**0.77**	**1.63**	**3.28**	**5.82**	**9.38**	**12.6**	**15.7**	**18.6**	**20.8**	21.67	**21.1**	**20.7**	**20.8**	**20.9**	**20.8**

Table 2. Measured GFLOPS for the CUFFT library and the proposed locality-improved FFT version (liFFT). N represents the number of coefficients of the transform for a batch of 8 signals ($b = 8$). Void entry corresponds to CUFFT unsupported configuration.

Batch of 8 signals, N coefficients per signal													
$\log_2(N)$	10	11	12	13	14	15	16	17	18	19	20	21	22
CUFFT	**3.44**	3.64	6.18	9.61	10.0	12.9	18.8	20.9	17.1	**23.3**	**26.2**	**22.5**	20.2
liFFT	2.80	**4.81**	**7.95**	**11.9**	**14.9**	**17.3**	**19.7**	**21.0**	**21.9**	22.7	22.6	21.5	**20.8**

Table 3. Measured GFLOPS for the CUFFT library and the proposed locality-improved FFT version (liFFT). N represents the number of coefficients of each signal in the batch. The total number of coefficients ($b \times N$) to be processed is fixed to 2^{20} coefficients.

2^{20} coefficients, batch of $2^{20}/N$ signals											
$\log_2(N)$	10	11	12	13	14	15	16	17	18	19	20
CUFFT	**39.9**	**21.2**	**19.6**	19.7	12.9	15.2	20.4	20.9	16.5	21.4	23.0
liFFT	17.9	18.6	19.3	**19.8**	**20.4**	**20.1**	**20.7**	**21.0**	**21.4**	**21.9**	21.7

Table 4. Measured GFLOPS for the CUFFT library and the proposed locality-improved FFT version (liFFT). N represents the number of coefficients of each signal in the batch. The total number of coefficients ($b \times N$) to be processed is fixed to 2^{25} coefficients. Void entries correspond to unsupported memory configurations.

2^{25} coefficients, batch of $2^{25}/N$ signals																
$\log_2(N)$	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
CUFFT	**43.8**	**25.1**	-	-	13.6	16.2	**22.0**	**22.7**	17.7	**23.8**	**26.5**	**22.7**	20.2	19.5	-	-
liFFT	19.0	19.8	**20.5**	**21.0**	**21.3**	**21.6**	21.8	22.1	**22.6**	22.9	22.6	21.4	**20.8**	**20.8**	**21.0**	**20.8**

1024-sample signals ($N = 1024$) can be permormed by one FFT call using an input vector of 4096 coefficients arranged in a 4-signal batch ($b = 4$). Observe that in this case only 10 FFT levels are carried out. Therefore the measured GFLOPS can be calculated as

$$GFLOPS = b\frac{5Nlog_2(N)}{t_{\mathrm{FFT}}}10^{-9}.$$

In the case of CUFFT library, the measured time includes function invocations to create the configuration plan and release its resources.

Tables 1, 2, 3 and 4 show the experimental results, measured in GFLOPS, by using the previous definition. Table 1 compiles results corresponding to single-signal tranforms in function of the signal size (from 1Ksamples to 32Msamples). The performance of the proposed locality-improved FFT implementation is compared with this one of the CUFFT library. A similar comparison is shown in Table 2, where 8-signal batch transforms are considered. Note that for our locality-improved implementation, the upper limit for the total number of coefficients ($b \times N$) is imposed by the size of the device memory, being 2^{25} coefficients. Results in tables 3 and 4 show, by means of two series of experiments, the effect of the number of signals in the batch. In both series, the total number of coefficients ($b \times N$) to be processed has been kept constant and equal to 2^{20} and 2^{25} respectively. Observe that, for 2^{25} coefficients, the CUFFT library does not support some configurations although all the coefficients fit in the global memory.

Although for batches with a large number of small signals, the CUFFT implementation appears to perform better, the proposed implementation makes a good exploitation of memory locality, allowing a good scalability with the signal size. In fact, for several interesting situations the locality-improved implementation

is able to provide better results than CUFFT. Examples of such situations are when there is a high number of signals in a batch and for very large signal size. This ability to manage large size signal constitutes an important feature of the proposed implementation. The CUFFT library is unable to perform the transform beyond 8 million elements (2^{23}) [8] whereas our implementation can manage up to 2^{25} coefficients (about 32 million samples), making a better exploitation of the available device memory.

5 Related Work

The FFT represents a computationally intensive floating-point algorithm whose generalized application makes it adequate for being accelerated on graphics platforms. Due to its interest, several contributions can be found in the literature of the last years focused on porting FFT algorithms to graphics processing units. In [9] very basic ideas of how to implement the FFT algorithm are collected. In [7], implementations of the FFT in the context of image processing applications are presented using GPU shader programming. Also in other specific contexts FFT has been developed on graphics hardware, like [10,1,5]. A discussion about the FFT implementation, together with other algorithms, is found in [4]. This last work tries to exploit the GPU memory hierarchy in order to improve the performance of the implementations but using programming models prior to CUDA.

6 Conclusions

Locality features of some implementations of the Fast Fourier Transform using the NVIDIA CUDA programming model are discussed in this work. A radix-two decimation-in-time FFT implementation is proposed, that can take advantage of the GPU memory organization. With this purpose, the proposed implementation intends to exploit memory reference locality, making an optimized use of the parallel data cache of the target device. Compared to the FFT library provided by the graphics processor manufacturer, our proposal exhibits a good scalability and it is able to achieve a better performance for certain signal sizes. Moreover, it is able to work with signals of larger size than the manufacturer's implementation.

References

1. Fialka, O., Cadik, M.: FFT and Convolution Performance in Image Filtering on GPU. Information Visualization (2006)
2. Fastest Fourier Transform in the West (FFTW), `http://www.fftw.org/`
3. Frigo, M., Johnson, S.G.: The Design and Implementation of FFTW3. Proceedings of the IEEE 93, 216–231 (2005)
4. Govindaraju, N.K., Larsen, S., Gray, J., Manocha, D.: A Memory Model for Scientific Algorithms on Graphics Processors. In: Conference on Supercomputing (2006)

5. Jansen, T., von Rymon-Lipinski, B., Hanssen, N., Keeve, E.: Fourier volume rendering on the GPU using a split-stream FFT. In: Vision, Modeling, and Visualization Workshop (2004)
6. Moler, C.: HPC Benchmark. In: Conference on Supercomputing (2006), http://www.hpcchallenge.org/presentations/sc2006/moler-slides.pdf
7. Moreland, K., Angel, E.: The FFT on a GPU. In: ACM Conference on Graphics Hardware (2003)
8. NVIDIA CUDA Homepage, http://developer.nvidia.com/object/cuda.html
9. Spitzer, J.: Implementing a GPU-Efficient FFT. SIGGRAPH GPGPU Course (2003)
10. Sumanaweera, T., Liu, D.: Medical Image Reconstruction with the FFT. GPU Gems 2, 765–784 (2005)

Large Eddy Simulation of Combustion on Massively Parallel Machines

Gabriel Staffelbach[1], Jean Mathieu Senoner[1], Laurent Gicquel[1], and Thierry Poinsot[2]

[1] CERFACS, 42 avenue Gaspard Coriolis, 31057 Toulouse Cedex 01, France
[2] IMFT (UMR CNRS-INPT-UPS 5502), Toulouse, France

Abstract. Combustion is the source of eighty percent of the energy produced in the world: it is therefore a topic of major interest in the present context of global warming and decreasing fuel resources. Simulating combustors and especially instability mechanisms in these systems has become a central issue in the combustion community in the last ten years. This can be achieved only on massively parallel computers. This paper presents modern techniques to simulate reacting flows in realistic geometries and shows how parallel computing (typically thousands of processors) has made these simulations possible. The physics of reacting flows are only discussed briefly to concentrate on specific issues linked to massively parallel computing, to the turbulent character of the flow and the effects of rounding errors.

1 Introduction

This paper presents an overview of combustion and of CFD (Computational Fluid Dynamics) for combustion. It focuses on the place of instabilities in reacting flows and on the role of massively parallel computations. These instabilities are found at many levels:

- Like any shear flow, reacting flows are submitted to hydrodynamic instabilities [1,2] and to vortex formation. Such vortices are easily observed in nature, like in the wake of aircrafts for example. When they are found within combustion chambers, they can constitute a major danger.
- Acoustics play a major role in reacting flows: by coupling with heat release, they are the source of a major problem in many combustion devices: combustion instabilities [3,4] which can induce high vibration levels and, in extreme cases, destroy combustion hardware in a few seconds.
- Instabilities are present in the physical problem to study but they are also present in the numerical methods used to simulate these mechanisms. Most high-fidelity numerical schemes required for Computational Fluid Dynamics exhibit low dissipation and therefore multiple non-physical instabilities (wiggles) which can require significant efforts to be kept under control [5,6,4].
- Finally, CFD for reacting flows is performed today on massively parallel machines: these architectures coupled with centered schemes needed for turbulent flows lead to an additional type of instability linked to the growth of

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 444–464, 2008.

rounding errors and to a new type of instability where the solution depends on unexpected parameters such as the commutativity errors of addition, the initial condition or the number of processors.

All these phenomena are 'instabilities' even though they correspond to very different physical mechanisms. In many cases, they can couple and in LES of combustion instabilities, the first issue is to be able to control the non physical waves due to the high-order spatial scheme as well as the rounding errors due to massively parallel computing. In this paper, the physics of combustion and of instabilities are briefly discussed before presenting a code used for LES of combustion by multiple groups and discussing one specific issue linked to the effect of rounding errors in simulations of turbulent flows.

2 Combustion: The Source of Our Energy

Combustion is the unknown heart of most present problems discussed everyday on global change and pollution issues. More than eighty percent of the energy produced on earth is obtained by burning some fossil fuel. This combustion can be produced by burning wood and producing a few Watts or by running 20 meter long industrial turbines producing 200 MWatts or more. The processes used for combustion can be simplified and non optimized like for wood combustion or highly technological like in reciprocating engines. This makes combustion the first contributor to our life style, our energy consumption and to the production of pollutants such as NO_x or CO_2. This also implies that controlling global change problems implies first to control combustion technologies since they are the major source of the problem and the first place to act. Considering that there is no real substitute for combustion at the moment in many applications (aircrafts, cars, energy production), it also means that the optimization of combustion processes is the most effective method to control global change.

The optimization of combustion is an ongoing work since 1900 but recent progress in this field has been tremendous. In the last twenty years, combustion devices have been optimized in terms of efficiency and pollution emissions to reach norms which were impossible to imagine before. This has been done by the introduction of electronic monitoring and control (especially for car engines) but also by a better understanding of combustion phenomena and an optimization of the parameters of combustion chambers. These parameters are not limited to the combustion chamber shape: the fuel injection strategy, for example, is a key point to control combustion. Optimizing a combustion chamber is therefore an extremely difficult process and this complexity is obvious when one considers the results of these optimization processes in combustion companies: while the shapes of most civil aircrafts today look the same, all combustion chambers are different showing that the optimum is by no means simple to define.

What makes combustor optimization even more difficult is the multiple non-linearities and instabilities found in reacting flows:

– Minimizing pollutant is easy to obtain by simply injecting less fuel in a chamber. The problem however is that, below a certain equivalence ratio

(below a certain amount of kg of fuel per kg of air), combustion simply stops [3,7,4]. The existence of this flammability limit makes optimization delicate because bringing the combustor close to extinction is dangerous (for aircrafts and helicopters for example, this is definitely something which must be avoided for obvious reasons).
- Optimization of combustion devices must be sought for a whole range of operating conditions. Many chambers can be optimized at one regime (for example idle conditions in a car) but then will not be efficient for another regime (full power for example). Moreover, a chamber can be optimized for a regime (a gas turbine for example) but impossible to ignite or too sensitive to sudden quenching.
- The most critical problem encountered since the end of the 20th century in the field of gas turbines is combustion instabilities [8,9,10,4]. Most chambers which were optimized to minimize NOx emissions and maximize efficiency in the last ten years have been subject to combustion instability problems. In Europe, the LOW NOx projects initiated by the European Commission are now being continued through combustion instability studies because the gains in NOx and efficiency are often compromised by the impact of combustion instabilities. Section 3 will focus on this specific issue.

3 Combustion and Instabilities

Reacting flows are compressible flows. They exhibit acoustic / combustion instabilities which can be extremely strong [3,11,4]. The fact that flames can couple with acoustics has been known for a long time [12] even though it is still not fully understood. Combustion instabilities are difficult to predict and are usually discovered at a late stage during the development of engine programmes so that they represent a significant industrial risk.

In steady combustors like gas turbines, instabilities can lead to oscillations of all flow parameters, reaching levels which are incompatible with the normal operation of the chamber. High levels of structure oscillations are found, very high levels of RMS pressure can be observed. In a given chamber, while normal turbulent combustion usually leads to 10 to 100 Pa RMS pressure levels, it is not uncommon to see chambers where the RMS pressure reaches 20000 Pa (180 dB) when a combustion instability begins. At these levels, the acoustic velocity associated to the RMS pressure can reach 1 to 20 m/s so that the perturbations induced by the acoustic field are absolutely not negligible. In such cases, the engine structure can fail, the fuel injector can burn, the flame might totally quench or flashback. Flashback is a phenomenon encountered when the acoustic velocity is larger than the mean flow leading to flow reversal at the combustor inlet: in other words, the flow leaves the combustor through the inlet instead of entering it; the flame does the same and ends up upstream of the combustion chamber, in a zone which was not designed to sustain high temperatures. Combustion instabilities have been the source of multiple failures in rocket engines, as early as the Saturne or the Ariane 4 project, in aircraft engines (main

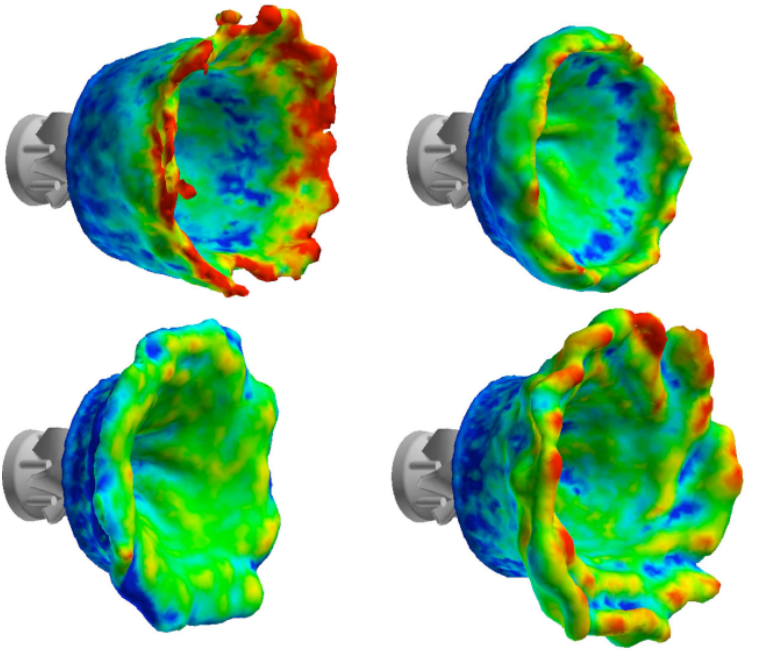


Fig. 1. Snapshots of flame position (isosurface of temperature) during one oscillation cycle at 120 Hz in an industrial gas turbine [13]. LES result

chamber of post combustion chamber), in industrial gas turbines, in industrial furnaces, etc. Fig. 1 shows an example of simulation of 'mild' oscillation in a gas turbine[13] where the flame position (visualized by an isosurface of temperature colored by axial velocity) pulsates strongly at four instants of a cycle occuring at 120 Hz). For such a mild oscillation, a limit cycle is obtained and the chamber can operate for a long time without problem except for a high noise level.

Predicting and controlling combustion instabilities is a major challenge for combustion research. Today, the most promising path is to understand these phenomena using Large Eddy Simulation methods which are able to predict these combustion oscillations [4,14,15] something which was impossible 10 years ago with classical Reynolds Averaged methods.

4 DNS, LES and RANS for Combustion

Turbulent combustion is encountered in most practical combustion systems such as rockets, internal combustion or aircraft engines, industrial burners and furnaces... while laminar combustion applications are almost limited to candles, lighters and some domestic furnaces. Studying and modeling turbulent combustion processes is therefore an important issue to develop and improve practical systems (*i.e.* to increase efficiency and reduce fuel consumption and pollutant

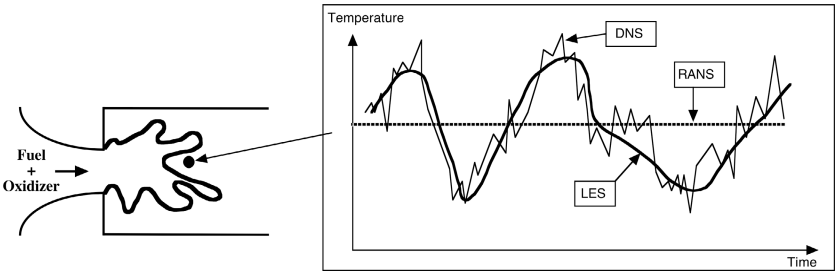


Fig. 2. Examples of time evolutions of the local temperature computed with DNS, RANS or LES in a turbulent flame brush

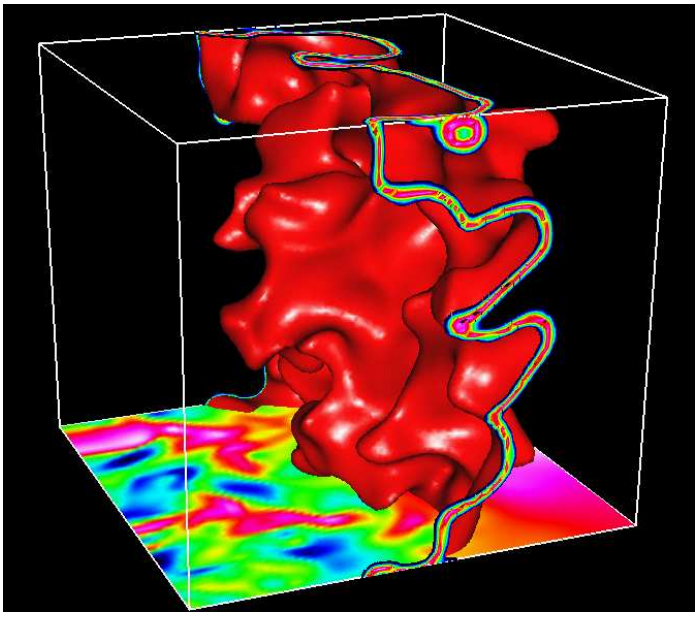


Fig. 3. DNS of a premixed flame interacting with three-dimensional isotropic turbulence [16]. An isosurface of temperature is visualized. The reaction rate is presented in two planes which are normal to the mean flame front. The vorticity field, corresponding to turbulent motions, is also displayed in the bottom plane.

formation). As combustion processes are difficult to handle using analytical techniques, numerical combustion for turbulent flames is a fast growing area.

The three main numerical approaches used in turbulent combustion modeling are Reynolds Averaged Navier Stokes (RANS) where all turbulent scales are modelled, Direct Numerical Simulation (DNS) where all scales are resolved and Large Eddy Simulation (LES) where larger scales are explicitly computed whereas the effects of smaller ones are modeled:

- Reynolds Averaged Navier Stokes (or RANS) computations have historically been the first possible approach because the computation of the instantaneous flow field in a turbulent flame was impossible. Therefore, RANS techniques were developed to solve for the mean values of all quantities. The balance equations for Reynolds or Favre (*i.e.* mass-weighted) averaged quantities are obtained by averaging the instantaneous balance equations. The averaged equations require closure rules: a turbulence model to deal with the flow dynamics in combination with a turbulent combustion model to describe chemical species conversion and heat release. Solving these equations provides averaged quantities corresponding to averages over time for stationary mean flows or averages over different realizations (or cycles) for periodic flows like those found in piston engines (*i.e.* phase averaging). For a stabilized flame, the temperature predicted with RANS at a given point is a constant corresponding to the mean temperature at this point (Fig. 2).
- The second level corresponds to Large-Eddy simulation (LES). The large vortices are explicitly calculated whereas the smaller ones are modeled using subgrid closure rules. The balance equations for large-eddy simulations are obtained by filtering the instantaneous balance equations. LES determine the instantaneous position of a "large scale" resolved flame front but a subgrid model is still required to take into account the effects of small turbulent scales on combustion. LES would capture the low-frequency variations of temperature (Fig. 2).
- The third level of combustion simulations is Direct Numerical Simulation (DNS) where the full instantaneous Navier-Stokes equations are solved without any model for turbulent motions: all turbulence scales are explicitly determined and their effects on combustion are captured. DNS would predict all time variations of temperature (Fig. 2) exactly like a high-resolution sensor would measure them in an experiment (Fig. 3). Developed in the last twenty years thanks to the development of high performance computers, DNS have changed the analysis of turbulent combustion but are still limited to simple academic flows (*i.e.* simple geometries and somewhat low Reynolds numbers).

In terms of computational requirements, CFD for non-reacting and reacting flows follow similar trends: DNS is the most demanding method and is limited to fairly low Reynolds numbers and simplified geometries. LES works with coarser grids (only larger scales have to be resolved) and may be used to deal with higher Reynolds numbers but require subgrid-scale models. In current engineering practice, RANS is extensively used because it is less demanding in terms of resources.

5 Massively Parallel LES of Combustion

Performing LES of real devices requires extremely large parallel machines. The solver used in this paper is an explicit solver called AVBP and developed by CERFACS and Institut Francais du Pétrole [17,18,19] for multiple industrial

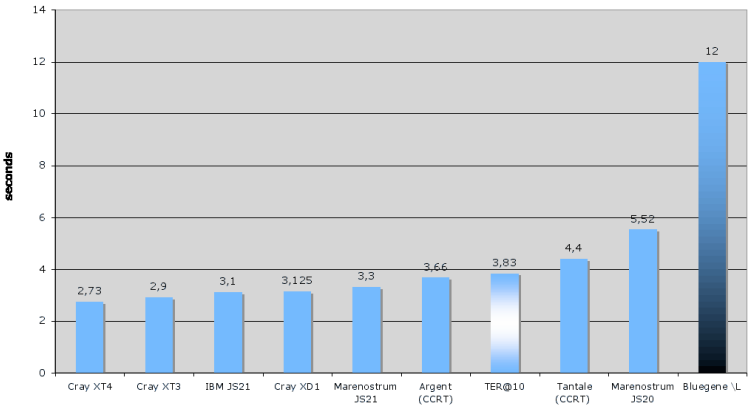


Fig. 4. Elapsed CPU times for AVBP to perform one time iteration on a 10 million point grid

users and research laboratories (www.cerfacs.fr/cfd/parallel.php). AVBP solves the compressible Navier Stokes equations in a multispecies gas with chemical reactions [4]. This implies typically advancing 10 variables (3 velocities, density, temperature and five chemical species) on a 10 million grid points over 1 million time iterations. The usual duration of a computation in physical time is of the order of 0.5 seconds for a real combustion chamber. This time is sufficient to understand instabilities which can develop in these devices. In terms of human time, such a computation can take from one day to one month.

Over the last ten years, AVBP has been applied succesfully to multiple non reacting flows [20,21,22], piston engine configurations [23,24], academic combustors [25,26,27,28], combustion instabilities [29,30,15], ramjet engines [31] and real gas turbines [32,33]. A key aspect of the success of AVBP is its capability to make use of all existing parallel machines with very limited adaptations.

To achieve efficiency on parallel architectures, AVBP uses MPI for message passing and HDF 5 for data format. Since the beginning of the AVBP project, two main choices have been made in order to be ready for massively parallel machines:

- LES solvers are unsteady solvers: they advance the solution in time. This is done with an explicit method to be able to scale on very large number of processors.
- The data structure is built for hybrid meshes (structured and unstructured) to allow easy mesh decomposition even on large number of processors.

As a result, AVBP has been used on multiple parallel architectures: Fig. 4 shows typical CPU times required to perform one time iteration on a 10 million cell grid using 32 processors. Of course, machines like BlueGene provide less efficient results when the number of processors is limited (here to 32). However, our objective is not to run on 32 processors but on 4000 or 40000. In this case, Fig. 5 shows that the parallel efficiency of machines like BlueGene is very high

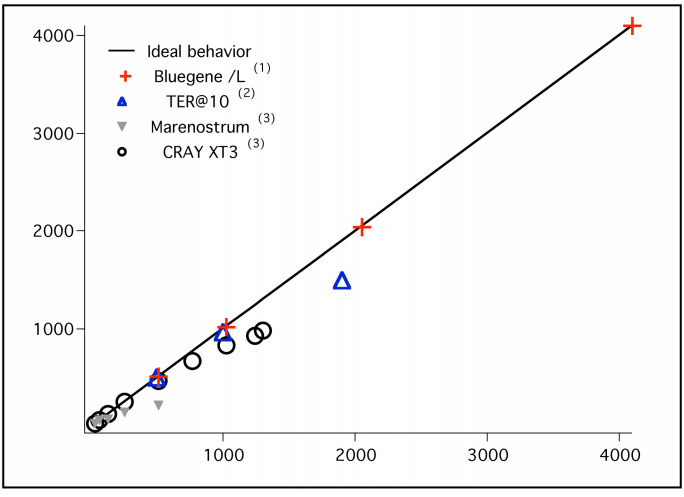


Fig. 5. Speed ups of AVBP measured on various parallel architectures

and that AVBP scales almost perfectly. This is a key issue when performing runs on machines with more than 100,000 processors (see for example the INCITE program at www.sc.doe.gov/ascr/incite/index.html)

6 Rounding Errors and LES

The literature shows the power of Large Eddy Simulation (LES) to predict non-reacting [34,35] as well as reacting turbulent flows[36,4,37,38,39,28]. The previous sections have suggested that the main strength of LES compared to classical Reynolds Averaged (RANS) methods is that, like Direct Numerical Simulation (DNS)[40,41,42], LES explicitely captures large scale unsteady motions due to turbulence instead of modeling them. An often ignored aspect of this feature is that like DNS, LES is also submitted to a well-known feature of turbulent flows: the exponential separation of trajectories [43] implies that the flow solution exhibited by LES is very sensitive to any "small perturbation" with respect to a reference state. These small perturbations which can induce new 'instabilities' can have different sources. Rounding errors are the first source of random noise in any finite precision computation: they constitute an unavoidable forcing for the Navier-Stokes equations and may lead to LES variability. The study of error growth in finite precision computations is an important topic in applied mathematics [44,45] but has found few applications in multidimensional fluid mechanics because of the complexity of the codes used in CFD.

Initial conditions are a second source of LES result variabilities: these conditions are often unknown and any small change in initial conditions may trigger significant changes in the LES solution.

Due to its large computational resource requirements, modern LES heavily relies on parallel computing. However, in codes using domain decomposition,

i.e. most of them, it is also an additional "noise" source in the Navier-Stokes equations especially at partition interfaces. Even in explicit codes where the algorithm is independent of the number of processors, the different summation orders with which a nodal value is reconstructed at partition interfaces may induce non-associativity errors. For example, in explicit codes on unstructured meshes using cell vertex methods[17], the residual at one node is obtained by adding the weighted residuals of the surrounding cells. Additions of only two summands are perfectly associative. Moreover, it must be noted that not all additions of more than two summands generate non-associativity errors. However, in some cases summation may yield distinct results for floating-point accumulation: the rounding errors in $(a+b)+c$ and in $a+(b+c)$ may be different, in particular if there are large differences in orders of magnitude between the terms[46]. After thousands of iterations, the LES result may be affected. Since these rounding errors are induced by non deterministic message arrival at partition interfaces, it is believed that such behaviour may occur for any unstructured parallel CFD code, regardless of the numerical schemes used. As a consequence, the simulation output might change when run on a different number of processors. The case of implicit codes in time[36,35,47] or in space such as compact schemes[48,49,50] is not considered here: for such schemes, the methods [51,52] used to solve the linear system appearing at each iteration depend on the number of processors. Therefore, rounding errors are not the only reason why solutions obtained with different numbers of processors differ. Even on a single processor computation, internal parameters of the partitioning algorithm may couple with rounding errors to force the LES solution. For example, a different reordering of nodes using the Cuthill-McKee (CM) or the reverse Cuthill-McKee (RCM) algorithm[53,54] may produce the same effect as a simple perturbation and can be the source of solution divergence.

Turbulence theory indicates that LES/DNS solutions have a meaning only in a statistical manner [55]: observing that the solution of a given LES/DNS at a given instant changes when the rounding errors or the initial conditions change is not really surprising. It becomes a real difficulty in the practical use of LES/DNS because running the same simulation on two different machines or one machine with a different number of processors or slightly different initial conditions can lead to totally different instantaneous results. For steady flows in the mean, statistics do not depend on these changes and mean profiles will be identical. However, when the objective of the LES is the study of unsteady phenomena such as ignition or quenching in a combustor[26], knowing that results depend on these parameters is certainly a sobering thought for the LES/DNS community and a drawback in terms of industrial exploitation.

This last section addresses these issues and tries to answer a simple question which is of interest for all practitioneers of LES: how does the solution produced by LES depend on the number of processors used to run the simulation? On the initial condition? On internal details of the algorithm?

First, we will present an example of the effects of the number of processors in a simple case: a rectangular turbulent channel computed with a fully explicit

LES code[19]. This example shows that even in an explicit code, running a simulation twice on a different number of processors can lead to totally different instantaneous solutions. The second section then gives a systematic description of the effects of rounding errors in two flows: a turbulent channel and a laminar Poiseuille flow. For all cases, the difference between two instantaneous solutions obtained by changing either the number of processors, the initial condition or the graph ordering is quantified in terms of norms between the two solutions. The effects of time step and machine precision (simple, double and quadruple) are also investigated.

6.1 Effects of the Number of Processors on LES

This first example is the LES of a rectangular fully developed turbulent channel of dimensions: 75x25x50 mm (Fig. 6). A pressure gradient is applied to a periodic channel flow and random disturbances are added to pass transition to turbulence. There are no boundary conditions except for the walls. The Reynolds number is $Re_\tau = \delta u_\tau/\nu = 1500$, where δ is half the channel height and u_τ the friction velocity at the wall: $u_\tau = (\tau_{wall}/\rho)^{1/2}$ with τ_{wall} being the wall stress. The mesh contains 30^3 hexahedral elements, it is not refined at walls. The first grid point is at a reduced distance $y^+ = yu_\tau/\nu \approx 100$ of the wall. The subgrid model is the Smagorinsky model and a law-of-the-wall is used at the walls[28]. The CFL number λ controlling the time step Δt is $\lambda = max((u + c)\Delta t/\Delta)$ where u is the local convective velocity, c the speed of sound and Δ the mesh size. For all simulations discussed below, the initial condition corresponds to a snapshot of the flow at a given instant, long after turbulence was initialized so that it is fully established. The computation is performed with an explicit code where domain decomposition is such that the method is perfectly equivalent on any number of processors. The Recursive Inertial Bisection (RIB)[56,57] algorithm has been used to partition the grid and the Cuthill-McKee algorithm is considered as the default graph reordering strategy. The scheme used here is the Lax-Wendroff scheme[58]. Additional tests were performed using a third-order scheme in space and time[18] but led to the same conclusions.

Figs. 7–9 show fields of axial velocity in the central plane of the channel at three instants after the run initialization. Two simulations performed on respectively 4 (TC1) and 8 processors (TC2) with identical initial conditions are compared. The characterictics of all presented simulations are displayed in Table 1 and 2. The instants correspond to (in wall units) $t^+ = 7.68$, $t^+ = 18.43$ and $t^+ = 26.11$ respectively where $t^+ = u_\tau t/\delta$. Obviously, the two flow fields observed at $t^+ = 7.68$ are identical. However, at $t^+ = 18.43$, differences start to become visible. Finally, at $t^+ = 26.11$, the instantaneous flow fields obtained in TC1 and TC2 are totally different. Even though the instantaneous flow fields are different, statistics remain the same: mean and root mean square axial velocity profiles averaged over $t^+ \approx 60$ are identical for both simulations.

This very simple example illustrates the main question of the present work: is the result of Figs. 7–9 reasonable? If it is not a simple programming error (the next section will show that it is not), can other parameters produce similar effects?

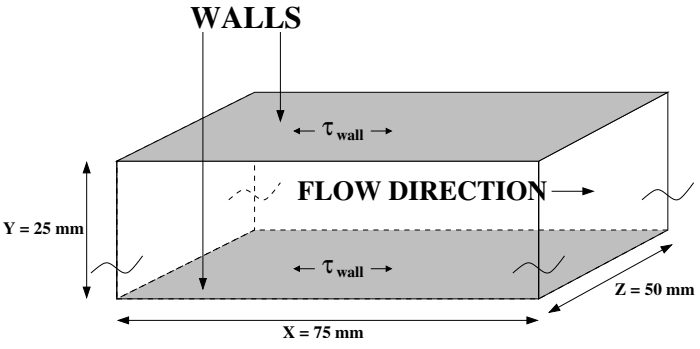
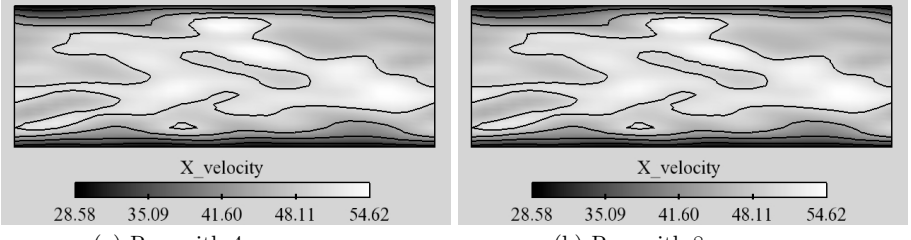
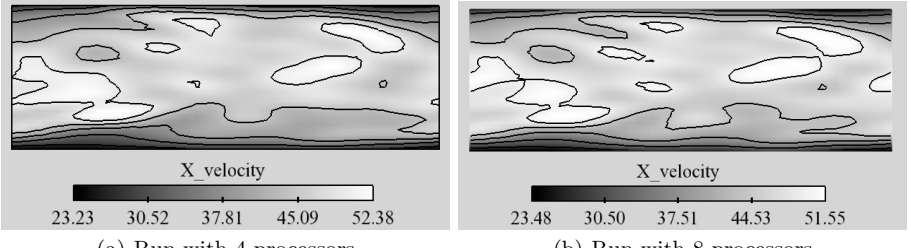


Fig. 6. Schematic of a periodic channel. The upper and lower boundaries consist of walls, all other boundaries are pairwise periodic.



(a) Run with 4 processors (b) Run with 8 processors

Fig. 7. Instantaneous field of axial velocity in the central plane of the channel at $t+ = 7.68$. a) run TC1 (4 processors), b) run TC2 (8 processors)



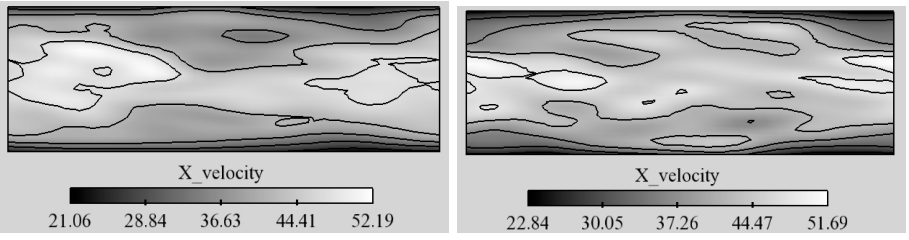
(a) Run with 4 processors (b) Run with 8 processors

Fig. 8. Instantaneous field of axial velocity in the central plane of the channel at $t+ = 18.43$. a) run TC1 (4 processors), b) run TC2 (8 processors)

6.2 Sensitivity of LES in Laminar and Turbulent Flows

To understand how LES can produce diverging instantaneous results such as those shown in the previous section, simple tests were performed to investigate the effects of various aspects of the methodology:

- laminar/turbulent baseline flow,
- number of processors,



(a) Run with 4 processors (b) Run with 8 processors

Fig. 9. Instantaneous field of axial velocity in the central plane of the channel at $t+ = 26.11$. a) run TC1 (4 processors), b) run TC2 (8 processors)

Table 1. Summary of turbulent LES runs (fully developed turbulent channel)

Run Id	Nbr proc	Init. cond.	Precision	Graph ordering	CFL λ
TC1	4	Fixed	Double	CM	0.7
TC2	8	Fixed	Double	CM	0.7
TC3	1	Fixed	Double	CM	0.7
TC4	1	Modif.	Double	CM	0.7
TC5	1	Fixed	Double	RCM	0.7
TC6	4	Fixed	Double	CM	0.35
TC7	8	Fixed	Double	CM	0.35
TC8	4	Fixed	Simple	CM	0.7
TC9	8	Fixed	Simple	CM	0.7
TC10	28	Fixed	Quadr.	CM	0.7
TC11	32	Fixed	Quadr.	CM	0.7

- initial condition,
- graph ordering,
- time step,
- machine precision.

For these tests, the objective is to quantify the differences between two LES solutions produced by a couple of simulations in Table 1 and 2. Let u_1 and u_2 be the scalar fields of two given instantaneous solutions at the same instant after initialization. A proper method to compare the latter is to use the following norms:

$$N_{max} = \max(u_1(\mathbf{x}) - u_2(\mathbf{x})) \text{ for } x \in \Omega \quad N_{mean} = \left(\frac{1}{V_\Omega}\int_\Omega (u_1(\mathbf{x}) - u_2(\mathbf{x}))^2 d\Omega\right)^{\frac{1}{2}} \text{ for } x \in \Omega \tag{1}$$

where Ω and V_Ω respectively denote the computational domain and its volume. Both norms (in m/s) will be applied to the axial velocity field so that N_{max} provides the maximum local velocity difference in the field between two solutions while N_{mean} yields a volumetrically averaged difference between the two solutions. The growth of N_{max} and N_{mean} versus the number of iterations will be used as a direct indicator for the divergence of the solutions.

6.3 A Fully Deterministic LES?

First, it is useful to indicate that performing any of the LES of Table 1 twice on the same machine with the same number of processors, the same initial conditions and the same partition algorithm leads to exactly the same solution, N_{max} and N_{mean} being zero to machine accuracy. In that sense, the LES remains fully deterministic. However, this is true only if the order of operations at interfaces is not determined by the order of message arrival so that summations are always carried out in the same order. Otherwise, the randomness induced by the non deterministic order of message arrival is enough to induce diverging solutions. Note that such an option can be expensive and that blocking messages order can increase the overall simulation cost by a large amount.

6.4 Influence of Turbulence

The first test is to compare a turbulent channel flow studied in the previous section and a laminar flow. A three dimensional Poiseuille flow was used as test case. The Poiseuille computation is performed on a pipe geometry with 361 by 26 points. The flow is laminar and the Reynolds number based on the bulk velocity and diameter is approximately 500. The boundary conditions are set periodic at the inlet/outlet and no slip at the duct walls, a constant axial pressure gradient is imposed in the entire domain.

Figure 10 shows the evolutions of N_{max} and N_{mean} versus iteration for runs TC1/TC2 and LP1/LP2. Note that the first point of the graph is the evaluation of the difference after one iteration. The only parameter tested here is a change of the number of processors. As expected from the snapshots of Figs. 7–9, the turbulent channel simulations are very sensitive to a change in the number of processors and the solutions of TC1 and TC2 diverge rapidly leading to a maximum difference of 20 m/s and a mean difference of 3-4 m/s after 90,000 iterations. On the other hand, the difference between LP1 and LP2 hardly increases and levels off when reaching values of the order or 10^{-12}. This is expected since there is obviously only one stable solution for the Poiseuille flow for infinite times and laminar flows do not induce exponential divergence of trajectories. However, this simple test case confirms that the turbulent character of the flow is the source of the divergence of solutions. This phenomenon must not be confused with the growth of a hydrodynamic mode, which is induced by the bifurcation in phase space of an equilibrium state of a given physical system. Obviously, such an equilibrium state does not exist for a fully developed turbulent channel flow.

Table 2. Summary of laminar runs (Poiseuille flow)

Run Id	Nbr proc	Init. cond.	Precision	Graph ordering	CFL λ
LP1	4	Fixed	Double	CM	0.7
LP2	8	Fixed	Double	CM	0.7

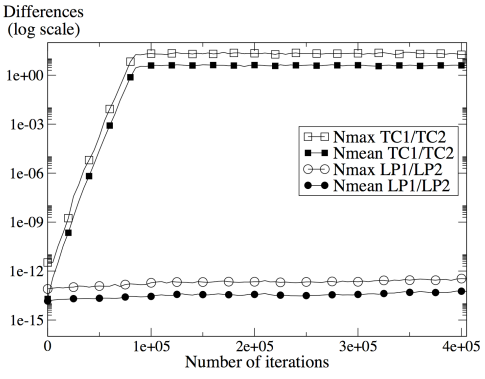


Fig. 10. Effects of turbulence. Differences between solutions measured by N_{max} (open symbols) and N_{mean} (closed symbols) versus iteration. Squares: differences between TC1 and TC2 (turbulent channel). Circles: differences between LP1 and LP2 (laminar Poiseuille flow).

In this case, the separation of trajectories is caused by vorticity, which leads to an increase in the number of degrees of freedom in phase space [59] and thus high sensitivity to initial conditions. Moreover, the stagnation of absolute and mean differences between TC1/TC2 simply implies that after 90,000 iterations solutions have become fully uncorrelated and should not be misinterpreted as the saturation of an exponentially growing mode.

The basic mechanism leading to Figs. 7–9 is that the turbulent flow acts as an amplifier for rounding errors generated by the fact that the mesh is decomposed differently in TC1 and TC2. The source of this difference is the new graph reordering obtained for both decompositions. This implies a different ordering when adding the contributions to a cell residual for nodes inside the subdomains but mainly at partition interfaces. This random noise roughly starts at machine accuracy (Fig. 10) at a few points in the flow and grows continuously if the flow is turbulent.

6.5 Influence of Initial Conditions

The previous section has shown that turbulence combined with a different domain decomposition (i.e. a different number of processors for the following) is sufficient to lead to totally different instantaneous flow realizations. It is expected that a perturbation in initial conditions will have the same effect as domain decomposition. This is verified in runs TC3 and TC4 which are run on one processor only, thereby eliminating issues linked to parallel implementation. The only difference between TC3 and TC4 is that in TC4, the initial solution is identical to TC3 except at one random point where a 10^{-16} perturbation is applied to the streamwise velocity component. Simulations with different locations of the perturbation were run to ensure that their position did not affect results.

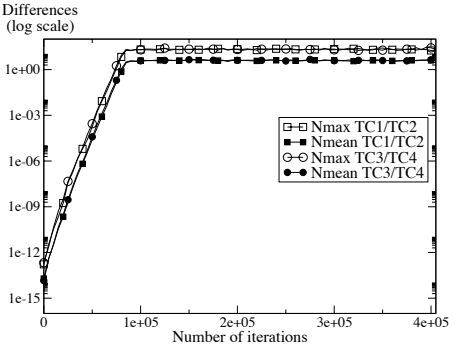


Fig. 11. Effects of initial conditions. Differences between solutions measured by N_{max} (open symbols) and N_{mean} (closed symbols) versus iteration. Squares: differences between TC1 and TC2 (different numbers of processors). Circles: differences between TC3 and TC4 (different initial conditions).

Figure 11 shows that the growth rate of the difference between TC3 and TC4 is exactly the same as the one observed between TC1 and TC2 (also displayed in Fig. 11): two solutions starting from a very slightly perturbed initial condition diverge as fast as two solutions starting from the same solution but running on different numbers of processors. Note that the difference between runs TC1 and TC2 comes from random rounding errors introduced at each time step while TC3 and TC4 differ only through the initial condition: no perturbation is added during the simulation. Still, the differences between TC3 and TC4 increase as fast as those between TC1 and TC2: this confirms that a turbulent flow amplifies any difference in the same manner, whether it is due to rounding errors or to a perturbation of the initial conditions.

6.6 Effects of Graph Ordering

It has already been indicated that performing the same simulation twice (with the same number of processors and same initial conditions) leads to exactly the same result. However, this is only true as long as exactly the same code is used. It is not verified any more as soon as a modification affecting rounding errors is done in the code. At this point, so many factors affecting rounding errors can be cited that a general discussion is pointless. This paper will focus on fully explicit codes and on one example only: the order used to add residuals at nodes in a cell vertex scheme. This order is controlled by the developer. For simulation TC5, the ordering of this addition was changed (reverse Cuthill-McKee algorithm): the residual at a given mesh node was assembled by adding the contributions to a cell residual in a different order. This change does not affect the flow data: in TC5 the node residual in a regular tetrahedral mesh is obtained by $1/4(R_1+(R_2+(R_3+R_4))$ where the R_i's are the residuals of the cells surrounding the node and by $1/4(R_4+(R_3+(R_2+R_1))$ in TC3. It has an effect, however, on rounding errors and the cumulated effects of this non-associativity

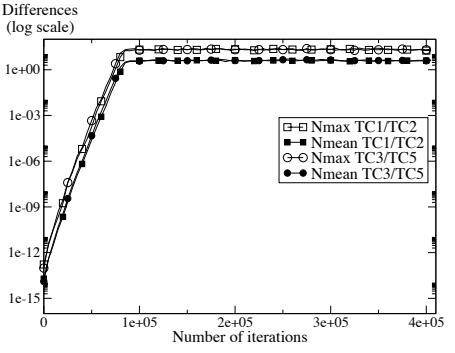


Fig. 12. Effects of addition order. Differences between solutions measured by N_{max} (open symbols) and N_{mean} (closed symbols) versus iteration. Squares: differences between TC1 and TC2. Circles: differences between TC3 and TC5.

error are what this test tries to isolate. TC5 and TC3 are performed with the same initial condition and run on one processor only. The only difference is the graph reordering strategy.

As shown by Fig. 12, the differences between TC5 and TC3 are again similar to those observed between TC1 and TC2 (obtained by changing the number of processors). This confirms that rounding errors (and not the parallel character of the code) are the source of the solution divergence. It also shows that any modification of the code could lead to such a divergence, suggesting that repeating an LES simulation with the same code after a few months and a few modifications will probably never yield the same instantaneous flow fields, potentially leading to discussions on the validity of the modified code.

6.7 Effects of Time Step

It is interesting to verify that numerical aspects do not influence the growth rate of the solutions difference and that the growth rate is only determined by the physical and geometrical parameters of the configuration. On that account, simulations TC6 and TC7 are performed with a time step reduced by a factor 2 compared to simulations TC1 and TC2. TC6 and TC7 are carried out on respectively 4 and 8 processors. The norms between TC6 and TC7 are displayed in Fig. 13 and compared to the norms between TC1 and TC2. From the explanations given above, similar growth rates are expected when comparing the growth rates over physical time. The growth rates observed in Fig. 13 are indeed very similar. The slight difference is probably due to the variation of the numerical dispersion and dissipation properties of the scheme with the time step [58].

6.8 Effects of Machine Precision

A last test to verify that the divergence between solutions is not due to a programming error but depends primarily on rounding errors is to perform the same

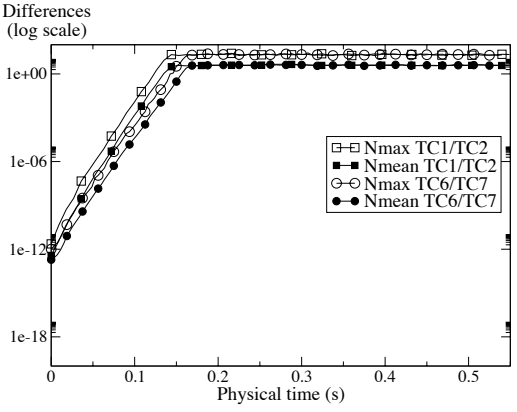


Fig. 13. Effects of time step. Differences between solutions measured by N_{max} (open symbols) and N_{mean} (closed symbols) versus physical time. Squares: differences between TC1 and TC2 (time step Δt). Circles: differences between TC6 and TC7 (time step $\Delta t/2$).

computation with simple/quadruple precision instead of double precision. Simulations TC1 and TC2 were repeated using simple precision in runs TC8 and TC9 (Table 1) and quadruple precision in TC10 and TC11. To compensate for the increase in computational time for quadruple precision simulations, roughly a factor ten compared to double precision, TC10 and TC11 were carried out on respectively 28 and 32 processors in order to yield a reasonable restitution time.

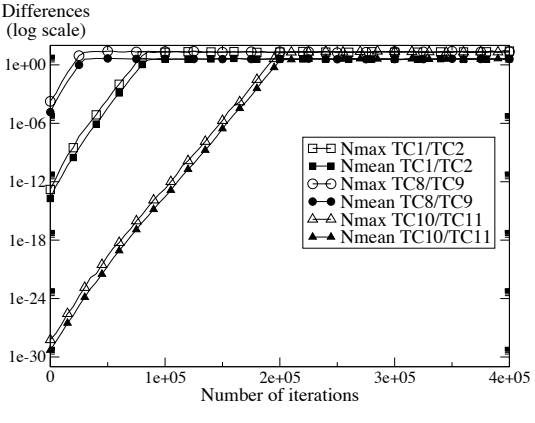


Fig. 14. Effects of machine accuracy. Differences between solutions measured by N_{max} (open symbols) and N_{mean} (closed symbols) versus iteration. Squares: differences between TC1 and TC2 (double precision). Circles: differences between TC8 and TC9 (simple precision). Triangles: differences between TC10 and TC11 (quadruple precision).

Results are displayed in Fig. 14 and compared to the difference between TC1 and TC2.

Figure 14 shows that the solutions differences for TC8/TC9 and TC10/TC11 roughly start from the respective machine accuracies (differences of 10^{-6} for simple precision after one iteration, differences of 10^{-30} for quadruple precision after one iteration) and increase exponentially with the same growth rate before reaching the same difference levels for all three cases. This shows that higher precision computations cannot prevent the exponentional divergence of trajectories but only delay it.

7 Conclusions

This paper has shown the power of Large Eddy Simulation to understand combustion instabilities and has focused on the efficiency of modern parallel solvers for reacting flows. A new specific question raised by these solvers for turbulent flows is the sensitivity of instantaneous LES fields to multiple parameters such as number of processors, initial condition, time step, changes in addition ordering of cell residuals for cell vertex methods. The baseline simulation used for the tests was a fully developed turbulent channel. Results confirm that any turbulent flow computed in LES exhibits significant sensitivity to these parameters, leading to instantaneous solutions which can be totally different. Laminar flows are almost insensitive to these parameters. The divergence of solutions is due to two combined facts: (1) the exponential separation of trajectories in turbulent flows and (2) the non-deterministic rounding errors induced by different domain decompositions or different ordering of operations. More generally any change in the code lines affecting rounding errors will have the same effects. Similarly, small changes in initial condition (of the order of machine accuracy at one point of the flow only) produce a divergence of solutions. Working with higher precision machines does not suppress the divergence of solutions but delays it.

These results confirm the expected nature of LES [55] in which solutions are meaningful only in a statistical sense and instantaneous values can not be used for analysis. However, on a more practical level, they point out various difficulties to develop LES codes: (1) repeating the results of a given LES after modifying the code and verifying that instantaneous solutions have not changed is not always possible. Since any programming error will also lead to a change in instantaneous solutions, identifying errors introduced by new lines will require a detailed analysis based on average fields (and not on instantaneous fields) and a significant loss of time. (2) Verifying a LES code on a parallel machine is a difficult task: running the code on different numbers of processors will lead to different solutions and make comparisons impossible. (3) Porting a LES code from one machine to another will also produce different solutions for turbulent runs, making comparison and validations of new architectures difficult.

The concept of "quality" in LES requires obviously more detailed studies and tools than what has been used up to now in Reynolds Averaged simulations. Instabilities appearing in a given LES on a given computer can have sources

which were not expected at first sight (like the number of processors). Mastering these instabilities (or at least understanding them) will be an important task to get the full power of LES techniques.

References

1. Drazin, P.G., Reid, W.H.: Hydrodynamic stability. Cambridge University Press, London (1981)
2. Ho, C.M., Huerre, P.: Perturbed free shear layers. J. Fluid Mech. 16, 365 (1984)
3. Williams, F.A.: Combustion theory. Benjamin Cummings, Menlo Park (1985)
4. Poinsot, T., Veynante, D.: Theoretical and numerical combustion. R.T. Edwards, 2nd edn. (2005)
5. Vichnevetsky, R., Bowles, J.B.: Fourier analysis of numerical approximations of hyperbolic equations. SIAM Studies in Applied Mechanics, Philadelphia (1982)
6. Sengupta, T.K.: Fundamentals of Computational Fluid Dynamics. Universities Press, Hyderabad, India (2004)
7. Kuo, K.K.: Principles of Combustion. John Wiley, New York (1986)
8. Candel, S.: Combustion instabilities coupled by pressure waves and their active control. In: 24th Symp (Int.) on Combustion, pp. 1277–1296. The Combustion Institute, Pittsburgh (1992)
9. McManus, K., Poinsot, T., Candel, S.: A review of active control of combustion instabilities. Prog. Energy Comb. Sci. 19, 1–29 (1993)
10. Lieuwen, T., Yang, V.: Combustion instabilities in gas turbine engines. operational experience, fundamental mechanisms and modeling. In: Prog. in Astronautics and Aeronautics AIAA, vol. 210 (2005)
11. Crighton, D.G., Dowling, A.P., Williams, J.E.F., Heckl, M., Leppington, F.: Modern methods in analytical acoustics. LNCS. Springer, Heidelberg (1992)
12. Rayleigh, L.: The explanation of certain acoustic phenomena. Nature 18, 319–321 (1878)
13. Giauque, A., Selle, L., Poinsot, T., Buechner, H., Kaufmann, P., Krebs, W.: System identification of a large-scale swirled partially premixed combustor using LES and measurements. J. Turb. 6(21), 1–20 (2005)
14. Martin, C., Benoit, L., Sommerer, Y., Nicoud, F., Poinsot, T.: Les and acoustic analysis of combustion instability in a staged turbulent swirled combustor. AIAA Journal 44(4), 741–750 (2006)
15. Selle, L., Benoit, L., Poinsot, T., Nicoud, F., Krebs, W.: Joint use of compressible large-eddy simulation and helmoltz solvers for the analysis of rotating modes in an industrial swirled burner. Combust. Flame 145(1-2), 194–205 (2006)
16. Boger, M., Veynante, D., Boughanem, H., Trouvé, A.: Direct numerical simulation analysis of flame surface density concept for large eddy simulation of turbulent premixed combustion. In: 27th Symp (Int.) on Combustion, Boulder, pp. 917–927. The Combustion Institute, Pittsburgh (1998)
17. Schönfeld, T., Rudgyard, M.: Steady and unsteady flows simulations using the hybrid flow solver avbp. AIAA Journal 37(11), 1378–1385 (1999)
18. Colin, O., Rudgyard, M.: Development of high-order taylor-galerkin schemes for unsteady calculations. J. Comput. Phys. 162(2), 338–371 (2000)
19. Moureau, V., Lartigue, G., Sommerer, Y., Angelberger, C., Colin, O., Poinsot, T.: High-order methods for DNS and LES of compressible multi-component reacting flows on fixed and moving grids. J. Comput. Phys. 202(2), 710–736 (2005)

20. Prière, C., Gicquel, L.Y.M., Kaufmann, A., Krebs, W., Poinsot, T.: Les of mixing enhancement: LES predictions of mixing enhancement for jets in cross-flows. J. Turb. 5, 1–30 (2004)
21. Prière, C., Gicquel, L.Y.M., Gajan, P., Strzelecki, A., Poinsot, T., Bérat, C.: Experimental and numerical studies of dilution systems for low emission combustors. Am. Inst. Aeronaut. Astronaut. J. 43(8), 1753–1766 (2005)
22. Mendez, S., Nicoud, F.: Large-eddy simulation of a bi-periodic turbulent flow with effusion. J. Fluid Mech. (in press, 2008)
23. Moureau, V.R., Vasilyev, O.V., Angelberger, C., Poinsot, T.J.: Commutation errors in Large Eddy Simulations on moving grids: Application to piston engine flows. In: Proc. of the Summer Program, Center for Turbulence Research, pp. 157–168. NASA AMES/Stanford University, USA (2004)
24. Richard, S., Colin, O., Vermorel, O., Benkenida, A., Angelberger, C., Veynante, D.: Towards large eddy simulation of combustion in spark ignition engines. Proc. Combust. Inst. 31, 3059–3066 (2007)
25. Selle, L., Lartigue, G., Poinsot, T., Koch, R., Schildmacher, K.U., Krebs, W., Prade, B., Kaufmann, P., Veynante, D.: Compressible large-eddy simulation of turbulent combustion in complex geometry on unstructured meshes. Combust. Flame 137(4), 489–505 (2004)
26. Sommerer, Y., Galley, D., Poinsot, T., Ducruix, S., Lacas, F., Veynante, D.: Large eddy simulation and experimental study of flashback and blow-off in a lean partially premixed swirled burner. J. Turb. 5 (2004)
27. Roux, S., Lartigue, G., Poinsot, T., Meier, U., Bérat, C.: Studies of mean and unsteady flow in a swirled combustor using experiments, acoustic analysis and large eddy simulations. Combust. Flame 141, 40–54 (2005)
28. Schmitt, P., Poinsot, T.J., Schuermans, B., Geigle, K.: Large-eddy simulation and experimental study of heat transfer, nitric oxide emissions and combustion instability in a swirled turbulent high pressure burner. J. Fluid Mech. 570, 17–46 (2007)
29. Kaufmann, A., Nicoud, F., Poinsot, T.: Flow forcing techniques for numerical simulation of combustion instabilities. Combust. Flame 131, 371–385 (2002)
30. Truffin, K., Poinsot, T.: Comparison and extension of methods for acoustic identification of burners. Combust. Flame 142(4), 388–400 (2005)
31. Roux, A., Gicquel, L.Y.M., Sommerer, Y., Poinsot, T.J.: Large eddy simulation of mean and oscillating flow in a side-dump ramjet combustor. Combust. Flame (in press, 2007)
32. Boudier, G., Gicquel, L.Y.M., Poinsot, T., Bissières, D., Bérat, C.: Comparison of LES, RANS and experiments in an aeronautical gas turbine combustion chamber. Proc. Combust. Inst. 31, 3075–3082 (2007)
33. Schildmacher, K.U., Hoffman, A., Selle, L., Koch, R., Schulz, C., Bauer, H.J., Poinsot, T.: Unsteady flame and flow field interaction of a premixed model gas turbine burner. Proc. Combust. Inst. 31, 3197–3205 (2007)
34. Sagaut, P.: Large eddy simulation for incompressible flows. Springer, Heidelberg (2002)
35. Mahesh, K., Constantinescu, G., Moin, P.: A numerical method for large-eddy simulation in complex geometries. J. Comput. Phys. 197(1), 215–240 (2004)
36. Mare, F.D., Jones, W.P., Menzies, K.: Large eddy simulation of a model gas turbine combustor. Combust. Flame 137, 278–295 (2001)
37. Pitsch, H.: Large eddy simulation of turbulent combustion. Ann. Rev. Fluid Mech. 38, 453–482 (2006)
38. El-Asrag, H., Menon, S.: Large eddy simulation of bluff-body stabilized swirling non-premixed flames. Proc. Combust. Inst. 31, 1747–1754 (2007)

39. Duwig, C., Fuchs, L., Griebel, P., Siewert, P., Boschek, E.: Study of a confined jurbulent jet: influence of combustion and pressure. AIAA Journal 45(3), 624–661 (2007)
40. Poinsot, T., Candel, S., Trouvé, A.: Application of direct numerical simulation to premixed turbulent combustion. Prog. Energy Comb. Sci. 21, 531–576 (1996)
41. Moin, P., Mahesh, K.: Dns: a tool in turbulence research. Ann. Rev. Fluid Mech. 30, 539–578 (1998)
42. Vervisch, L., Poinsot, T.: Direct numerical simulation of non premixed turbulent flames. Ann. Rev. Fluid Mech. 30, 655–692 (1998)
43. Tennekes, H., Lumley, J.L.: A first course in turbulence. MIT Press, Cambridge (1972)
44. Stoer, J.S., Bulirsch, R.: An introduction to numerical analysis. Springer, Berlin (1980)
45. Chaitin-Chatelin, F., Frayssé, V.: Lectures on Finite Precision Computations. SIAM, Philadelphia (1996)
46. Hanrot, G., Lefèvre, V., Stehlé, D., Zimmermann, P.: Worst cases for a periodic function with large arguments. In: Kornerup, P., Muller, J.M. (eds.) Proceedings of the 18th IEEE Symposium on Computer Arithmetic, pp. 133–140. IEEE Computer Society Press, Los Alamitos (2007)
47. Freitag, M., Janicka, J.: Investigation of a strongly swirled premixed flame using les. Proc. Combust. Inst. 31, 1477–1485 (2007)
48. Lele, S.: Compact finite difference schemes with spectral like resolution. J. Comput. Phys. 103, 16–42 (1992)
49. Abarbanel, S.S., Chertock, A.E.: Strict stability of high-order compact implicit finite-difference schemes: the role of boundary conditions for hyperbolic PDEs, I. J. Comput. Phys. 160, 42–66 (2000)
50. Sengupta, T.K., Ganerwal, G., Dipankar, A.: High accuracy compact schemes and gibbs' phenomenon. J. Sci. Comput. 21(3), 253–268 (2004)
51. Saad, Y.: A flexible inner-outer preconditioned gmres algorithm. SIAM J. Sci. Comput. 14, 461–469 (1993)
52. Fraysse, V., Giraud, L., Gratton, S.: A set of flexible-gmres routines for real and complex arithmetics. Technical Report TR/PA/98/20, CERFACS (1998)
53. Cuthill, E., McKee, J.: Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the 24th National Conference of the ACM, pp. 157–172 (1969)
54. Liu, W.H., Sherman, A.H.: Comparative analysis of the cuthill-mckee and the reverse cuthill-mckee ordering algorithms for sparse matrices. SIAM Journal of Numerical Analysis 13(2), 198–213 (1976)
55. Pope, S.B.: Ten questions concerning the large-eddy simulation of turbulent flows. New Journal of Physics 6, 35 (2004)
56. Williams, R.D.: Performance of dynamic load balancing algorithms for unstructured mesh calculations. Concurrency: Practice, and Experience 3(5), 451–481 (1991)
57. Taylor, V.E., Nour-Omid, B.: A study of the factorization fill-in for a parallel implementation of the finite element method. Int. J. Numer. Meth. Eng. 37, 3809–3823 (1994)
58. Hirsch, C.: Numerical Computation of internal and external flows. John Wiley, New York (1988)
59. Aref, H.: Integrable, chaotic and turbulent vortex motion in two-dimensional flows. Ann. Rev. Fluid Mech. 15, 345–389 (1983)

Variational Multiscale LES and Hybrid RANS/LES Parallel Simulation of Complex Unsteady Flows

Hilde Ouvrard[1], Bruno Koobus[1], Maria-Vittoria Salvetti[2], Simone Camarri[2], and Alain Dervieux[3]

[1] Département de Mathématiques, Université Montpellier II, Place E. Bataillon, 34095 Montpellier, France
{houvrard,koobus}@math.univ-montp2.fr
[2] Dipartimento di Ingegneria Aerospaziale, Università di Pisa, Via G. Caruso, 56122 Pisa, Italy
{mv.salvetti,s.camarri}@ing.unipi.it
[3] INRIA, 2004 Route des Lucioles, 06902 Sophia-Antipolis, France
Alain.Dervieux@sophia.inria.fr

Abstract. We study a new hybrid RANS/Variational Multiscale LES (VMS-LES) model for bluff body flows. The simulations have been carried out using a parallelized solver, based on a mixed element/volume formulation on unstructured grids. First, the behavior of a VMS-LES model with different subgrid scale models is investigated for the flow past a circular cylinder at Reynolds number Re =3900. Second, a new strategy for blending RANS and LES methods in a hybrid model is described and applied to the simulation of the flow around a circular cylinder at $Re = 140000$.

Keywords: Bluff-body flows, variational multiscale LES, hybrid RANS/LES approach, parallel simulation.

1 Introduction

The approach involving the Reynolds-Averaged Navier-Stokes equations (RANS) is widely used for the simulation of complex turbulent flows. However these models are not sufficient to properly simulate complex flows with massive separations such as the flow around bluff bodies. The LES approach gives generally more accurate predictions but requires higher computational cost.

The traditional LES approach is based on a filtering operation, the large energy-containing scales are resolved and the smallest scales are modeled using a sub-grid scale (SGS) model. Usual LES subgrid stress modeling like in the Smagorinsky model are based on the assumption of an universal behavior of the subgrid scales. Due to this assumption, energy-containing eddies must not be filtered. Then large Reynolds numbers cannot be addressed with reasonable coarse meshes, except, in particular regions of detached eddies. Even in the case of low Reynolds number or detached eddies, a particular attention must be paid

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 465–478, 2008.

to energetic eddies. For example, the classical eddy-viscosity models are purely dissipative. Often unable to model backscatter, they apply, instead, damping to large resolved energetic eddies.

Starting from these remarks, we investigate the application of a complementary mechanism for filtering scales. The Variational Multiscale (VMS) concept of Hughes [9] appears as a reasonable answer to the filtering issue. The VMS approach was originally introduced by Hughes [9,29] for the LES of incompressible flows and implemented in a Fourier spectral framework using a frequency cutoff for the scale separation (small and large scales). In this approach, the Navier-Stokes equations are not filtered but are treated by variational projection, and the effect of the unresolved scales is modeled only in the equations representing the small resolved scales. The VMS-LES approach (even with simple subgrid scale models as Smagorinsky's model) and dynamic LES models have shown similar order of accuracy, but the former is less computationally expensive and does not require any *ad hoc* treatement (smoothing and clipping of the dynamic constant, as usually required with dynamic LES models) in order to avoid stability problems. In this work, we consider the VMS-LES implementation presented in [13] for the simulation of compressible turbulent flows on unstructured grids within a mixed finite volume/finite element framwork. We investigate the effect of subgrid scale models in our VMS-LES method for the simulation of a bluff-body flow.

Another major difficulty for the success of LES for the simulation of complex flows is the fact that the cost of LES increases as the flow Reynolds number is increased. Indeed, the grid has to be fine enough to resolve a significant part of the turbulent scales, and this becomes particularly critical in the near-wall regions. A new class of models has recently been proposed in the literature in which RANS and LES approaches are combined together in order to obtain simulations as accurate as in the LES case but at reasonable computational costs. Among these so-called hybrid models described in the literature, the Detached Eddy Simulation (DES) [27] has received the largest attention. Among these so-called hybrid models, we proposed a new strategy for blending RANS and LES approaches in a hybrid model [20,7]. To this purpose, as in [16], the flow variables are decomposed in a RANS part (i.e. the averaged flow field), a correction part that takes into account the turbulent large-scale fluctuations, and a third part made of the unresolved or SGS fluctuations. The basic idea is to solve the RANS equations in the whole computational domain and to correct the obtained averaged flow field by adding, where the grid is adequately refined, the remaining resolved fluctuations. We search here for a hybridization strategy in which the RANS and LES models are blended in the computational domain following a given criterion. To this aim, a blending function is introduced, θ, which smoothly varies between 0 and 1. In particular, two different definitions of the blending function θ are proposed and examined in this paper. They are based on the ratios between (i) two eddy viscosities and (ii) two characteristic length scales. The RANS model used in the proposed hybrid approach is a

low-Reynolds version [8] of the standard $k-\varepsilon$ model, while for the LES part the Variational Multiscale approach (VMS) is adopted [9].

In this paper, we present VMS-LES and RANS/LES parallel simulations of bluff-body flows, by a computational fluid dynamics (CFD) software which combines mesh partitioning techniques and a domain decomposition method. These simulations require to discretize the fluid equations on large three-dimensional meshes with small time-steps. Therefore they require intensive computational ressource (in terms of CPU and memory) and parallel computation is of particular interest for such applications.

2 Turbulence Modeling

2.1 Variational Multiscale LES

In this paper, we consider the *Koobus-Farhat VMS implementation* [13] for the simulation of compressible turbulent flows. It uses the flow variable decomposition [6]:

$$W = \underbrace{\overline{W}}_{LRS} + \underbrace{W'}_{SRS} + W^{SGS} \tag{1}$$

where $\overline{W}$ is the large resolved scale (LRS) component of W, W' is its small resolved scale (SRS) component, and W^{SGS} the non-resolved component. The decomposition of the resolved component is obtained by projection onto two complementary spaces $\overline{\mathcal{W}}$ (LRS space) and $\mathcal{W}'$ (SRS space) of the resolved scale space:

$$\overline{W} \in \overline{\mathcal{W}} \quad ; \quad W' \in \mathcal{W}' \ . \tag{2}$$

A projector operator onto the LRS space $\overline{\mathcal{W}}$ is defined by spatial averaging on macro cells, obtained by finite-volume agglomeration which splits the basis/test functions ϕ_l into large scale basis denoted $\overline{\phi}_l$, and small scale basis denoted ϕ'_l.

$$\overline{W} = \sum \overline{W}_l \overline{\phi}_l \quad ; \quad W' = \sum W'_l \phi'_l \tag{3}$$

By variational projection onto $\overline{\mathcal{W}}$ and $\mathcal{W}'$, we obtain the equations governing the large resolved scales and the equations governing the small resolved scales. A key feature of the VMS model is that we set to zero the modeled influence of the unresolved scales on the large resolved ones. The SGS model is introduced only in the equations governing the small resolved scales, and, by combining the small and large resolved scale equations, the resulting Galerkin variational formulation of the VMS model writes:

$$\left(\frac{\partial(\overline{W}+W')}{\partial t}, \phi_l\right) + \left(\nabla \cdot F(\overline{W}+W'), \phi_l\right) = -\left(\ \tau^{LES}(W'), \phi'_l\right) \quad l = 1, N \tag{4}$$

where $\tau^{LES}(W')$ is the subgrid scale tensor computed using the SRS component and which is defined by a SGS eddy-viscosity model.

For the purpose of this study, three SGS eddy-viscosity models are considered: the classical model of Smagorinsky [25], and two recent and promising models, namely the WALE model [21] and the one of Vreman [31]. More details on this VMS-LES approach can be found in [13].

2.2 Hybrid RANS/VMS-LES

As in Labourasse and Sagaut [16], the following decomposition of the flow variables is adopted:

$$W = \underbrace{< W >}_{RANS} + \underbrace{W^c}_{correction} + W^{SGS}$$

where $< W >$ are the RANS flow variables, obtained by applying an averaging operator to the Navier-Stokes equations, W^c are the remaining resolved fluctuations (i.e. $< W > + W^c$ are the flow variables in LES) and W^{SGS} are the unresolved or SGS fluctuations.

If we write the Navier-Stokes equations in the following compact conservative form:

$$\frac{\partial W}{\partial t} + \nabla \cdot F(W) = 0$$

in which F represents both the viscous and the convective fluxes, for the averaged flow $\langle W \rangle$ we get:

$$\frac{\partial \langle W \rangle}{\partial t} + \nabla \cdot F(\langle W \rangle) = -\tau^{RANS}(\langle W \rangle) \tag{5}$$

where $\tau^{RANS}(\langle W \rangle)$ is the closure term given by a RANS turbulence model.

As well known, by applying a filtering operator to the Navier-Stokes equations, the LES equations are obtained, which can be written as follows:

$$\frac{\partial \langle W \rangle + W^c}{\partial t} + \nabla \cdot F(\langle W \rangle + W^c) = -\tau^{LES}(\langle W \rangle + W^c) \tag{6}$$

where τ^{LES} is the SGS term.

An equation for the resolved fluctuations W^c can thus be derived (see also [16]):

$$\frac{\partial W^c}{\partial t} + \nabla \cdot F(\langle W \rangle + W^c) \; - \; \nabla \cdot F(\langle W \rangle) = \tau^{RANS}(\langle W \rangle) - \tau^{LES}(\langle W \rangle + W^c) \tag{7}$$

The basic idea of the proposed hybrid model is to solve Eq. (5) in the whole domain and to correct the obtained averaged flow by adding the remaining resolved fluctuations (computed through Eq. (7)), wherever the grid resolution is adequate for a LES. To identify the regions where the additional fluctuations must be computed, we introduce a *blending function*, θ, smoothly varying between 0 and 1. When $\theta = 1$, no correction to $\langle W \rangle$ is computed and, thus, the RANS approach is recovered. Conversely, wherever $\theta < 1$, additional resolved

fluctuations are computed; in the limit of $\theta \to 0$ we want to recover a full LES approach. Thus, the following equation is used here for the correction term:

$$\begin{aligned} &\frac{\partial W^c}{\partial t} + \nabla \cdot F(\langle W \rangle + W^c) - \nabla \cdot F(\langle W \rangle) = \\ &(1-\theta)\left[\tau^{RANS}(\langle W \rangle) - \tau^{LES}(\langle W \rangle + W^c)\right] \end{aligned} \tag{8}$$

Although it could seem rather arbitrary from a physical point of view, in Eq. (8) the damping of the righthandside term through multiplication by $(1-\theta)$ is aimed to obtain a smooth transition between RANS and LES. More specifically, we wish to obtain a progressive addition of fluctuations when the grid resolution increases and the model switches from the RANS to the LES mode.

Summarizing, the ingredients of the proposed approach are: a RANS closure model, a SGS model for LES and the definition of the blending function.

RANS and LES closures: For the LES mode, we wish to recover the variational multiscale approach described in Section 2.1. Thus, the Galerkin projection of the equations for averaged flow and for correction terms in the proposed hybrid model become respectively:

$$\begin{aligned} &\left(\frac{\partial \langle W \rangle}{\partial t}, \psi_l\right) + (\nabla \cdot F_c(\langle W \rangle), \psi_l) + (\nabla \cdot F_v(\langle W \rangle), \phi_l) = \\ &-\left(\tau^{RANS}(\langle W \rangle), \phi_l\right) \quad l = 1, N \end{aligned} \tag{9}$$

$$\begin{aligned} &\left(\frac{\partial W^c}{\partial t}, \psi_l\right) + (\nabla \cdot F_c(\langle W \rangle + W^c), \psi_l) - (\nabla \cdot F_c(\langle W \rangle), \psi_l) + \\ &(\nabla \cdot F_v(W^c), \phi_l) = (1-\theta)\left[\left(\tau^{RANS}(\langle W \rangle), \phi_l\right) - \left(\tau^{LES}(W'), \phi_l'\right)\right] \quad l = 1, N \end{aligned} \tag{10}$$

where $\tau^{RANS}(\langle W \rangle$ is the closure term given by a RANS turbulence model and $\tau^{LES}(W')$ is given by one of the SGS closures mentioned in Section 2.1.

As far the closure of the RANS equations is concerned, the low Reynolds $k-\varepsilon$ model proposed in [8] is used.

Definition of the blending function and simplified model: As a possible choice for θ, the following function is used in the present study:

$$\theta = F(\xi) = tanh(\xi^2) \tag{11}$$

where ξ is the *blending parameter*, which should indicate whether the grid resolution is fine enough to resolve a significant part of the turbulence fluctuations, i.e. to obtain a LES-like simulation. The choice of the *blending parameter* is clearly a key point for the definition of the present hybrid model. In the present study, different options are proposed and investigated, namely: the ratio between the eddy viscosities given by the LES and the RANS closures and the ratio between the LES filter width and a typical length in the RANS approach.

To avoid the solution of two different systems of PDEs and the consequent increase of required computational resources, Eqs. (9) and (10) can be recast together as:

$$\left(\frac{\partial W}{\partial t}, \psi_l\right) + (\nabla \cdot F_c(W), \psi_l) + (\nabla \cdot F_v(W), \phi_l) = \\ -\theta \left(\tau^{RANS}(\langle W \rangle), \phi_l\right) - (1-\theta)\left(\tau^{LES}(W'), \phi_l'\right) \quad l = 1, N \tag{12}$$

Clearly, if only Eq. (12) is solved, $\langle W \rangle$ is not available at each time step. Two different options are possible: either to use an approximation of $\langle W \rangle$ obtained by averaging and smoothing of W, in the spirit of VMS, or to simply use in Eq. (12) $\tau^{RANS}(W)$. The second option is adopted here as a first approximation. We refer to [20,7] for further details.

3 Numerical Method and Parallelisation Strategy

The fluid solver AERO under consideration is based on a mixed finite element/finite volume formulation on unstructured tetrahedral meshes. The scheme is vertex-centered, the diffusive terms are discretized using P1 Galerkin finite elements and the convective terms with finite volumes. The Monotone Upwind Scheme for Conservation Laws reconstruction method (MUSCL) is adopted here and the scheme is stabilized with sixth-order spatial derivatives. An upwind parameter γ, which multiplies the stabilization part of the scheme, allows a direct control of the numerical viscosity, leading to a full upwind scheme for $\gamma = 1$ and to a centered scheme for $\gamma = 0$. This low-diffusion MUSCL reconstruction, which limits as far as possible the interaction between numerical and SGS dissipation, is described in detail in [4].

The flow equations are advanced in time with an implicit scheme, based on a second-order time-accurate backward difference scheme. The non-linear discretised equations are solved by a defect-correction (Newton-like) method in which a first order semi-discretisation of the Jacobian is used. At each time-step, the resulting sparse linear system is solved by a Restricted Additive Schwarz (RAS) method [13]. More specifically, the linear solver is based on GMRES with a RAS preconditioning and the subdomain problems are solved with ILU(0). Typically, two defect-correction iterations requiring each of them a maximum of 20 RAS iterations are used per time-step. This implicit scheme is linearly unconditionally stable and second-order accurate.

For what concerns the parallelisation strategy used in this study, it combines mesh partitioning techniques and a message-passing programming model [16,20]. The mesh is assumed to be partitioned into several submeshes, each one defining a subdomain. Basically the same serial code is going to be executed within every subdomain. Modifications for parallel implementation occured in the main stepping-loop in order to take into account several assembly phases of the subdomain results, depending on the fluid equations (viscous/inviscid flows), the spatial approximation and on the nature of the time advancing procedure (explcit/implicit). Because mesh partitions with overlapping incur redundant floating-point operations, non-overlapping mesh partitions are chosen. It has been shown in [20] that the latter option is more efficient though it induces additional communication steps. For our applications, in a prepocessing step we

use an automatic mesh partitioner that creates load balanced submeshes inducing a minimum amount of interprocessor communications. Data communications between neighboring subdomains are achieved through the MPI communication library.

For the simulations presented in the next section, the Roe-Turkel solver is used with a numerical viscosity parameter γ equal to 0.2. The CFL number was chosen so that a vortex shedding cycle is sampled in around 400 time steps for the low-Reynolds simulations and at least 600 time steps for the simulations at Re =140000.

4 VMS-LES Simulations

In this section, we apply our VMS-LES methodology to the simulation of a flow past a circular cylinder at Mach number $M_\infty = 0.1$. The subcritical Reynolds number equal to 3900, is based on body diameter and freestream velocity.

The computational domain size is: $-10 \leq x/D \leq 25$, $-20 \leq y/D \leq 20$ and $-\pi/2 \leq z/D \leq \pi/2$, where x, y and z denote the streamwise, transverse and spanwise direction respectively. The cylinder of unit diameter D is centered on $(x, y) = (0, 0)$.

The flow domain is discretized by an unstructured tetrahedral grid which consists of approximatively 2.9×10^5 nodes. The averaged distance of the nearest point to the cylinder boundary is $0.017D$, which corresponds to $y^+ \approx 3.31$.

For the purpose of these simulations, the Steger-Warming conditions [28] are imposed at the inflow and outflow as well as on the upper and lower surface ($y = \pm H_y$). In the spanwise direction periodic boundary conditions are applied and on the cylinder surface no-slip boundary conditions are set.

To investigate the effect of the different SGS models in the VMS-LES approach, three simulations are carried out on the same grid: VMS-LES with classical Smagorinsky's SGS model (VMSLES1), VMS-LES with Vreman's SGS model (VMSLES2) and VMS-LES with WALE SGS model (VMSLES3). To evaluate the performance of VMS-LES methodology over classical LES, three simulations using Smagorinsky's model (LES1), Vreman's model (LES2) and WALE model (LES3) are presented.

Instantaneous streamwise velocity isocontours using VMS-LES approach are plotted in Figure 1. This plot highlights the small structures predicted in the wake by the rather coarse mesh employed in this work, and the three dimensionality of the flow.

The averaged data are obtained using about 20 shedding cycles or 150 nondimensional time units after the initial transient period.

Time-averaged values and turbulence parameters are summarized in Table 1 and compared to data from experiments and to numerical results obtained by other investigators using dynamic LES models on finer grids (containing between half a million and 7.5 million nodes). Parameters obtained from VMS-LES simulations are in better agreement with the experimental values compared to

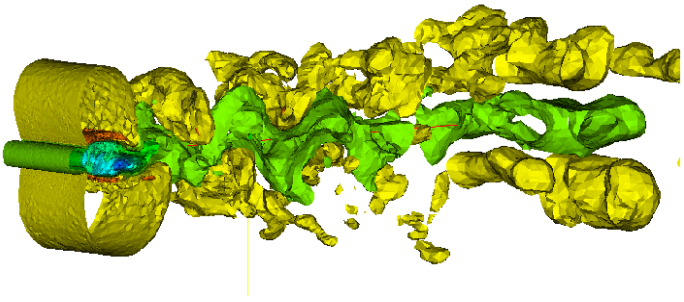


Fig. 1. Circular cylinder: Instantaneous streamwise velocity

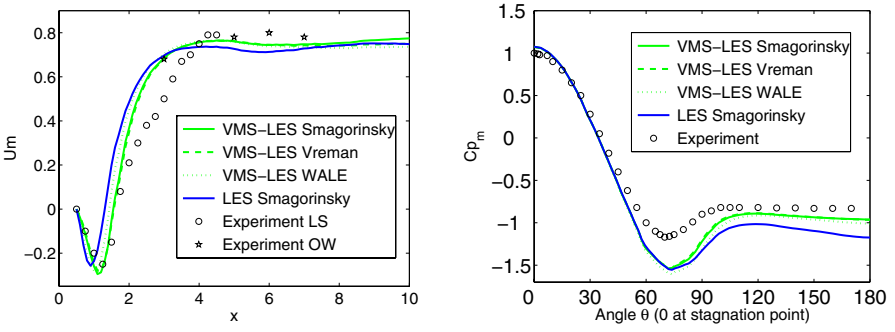
Table 1. Circular cylinder: Bulk coefficients, comparison with experimental data and with other simulations in the literature. $\overline{C_d}$ denotes the mean drag coefficient, St the Strouhal number, l_r the mean recirculation length: the distance on the centerline direction from the surface of the cylinder tot he point where the time-averaged streamwise velocity is zero, $\overline{C_{P_b}}$ the mean back-pressure coefficient and U_{min} the minimum centerline streamwise velocity.

data from	$\overline{C_d}$	St	l_r	$\overline{C_{P_b}}$	U_{min}
LES1	**1.16**	**0.212**	**0.81**	**-1.17**	**-0.26**
LES2	**1.04**	**0.221**	**0.97**	**-1.01**	**-0.28**
LES3	**1.14**	**0.214**	**0.75**	**-1.20**	**-0.25**
VMSLES1	**1.00**	**0.221**	**1.05**	**-0.96**	**-0.29**
VMSLES2	**1.00**	**0.220**	**1.07**	**-0.97**	**-0.28**
VMSLES3	**1.03**	**0.219**	**0.94**	**-1.01**	**-0.28**
Numerical data					
[15]	1.04	0.210	1.35	-0.94	-0.37
[3]	1.07		1.197	-1.011	
[10]	0.99	0.212	1.36	-0.94	-0.33
Experiments					
[22]	0.99±0.05	0.215±0.05		-0.88±0.05	-0.24±0.1
[5]		0.215±0.005	1.33±0.05		
[23]		0.21±0.005	1.4±0.1		-0.24±0.1
[19]			1.18±0.05		

simulations using LES, especially for the prediction of the mean drag. Smagorinsky's and Vreman's SGS model give best results.

Figure 2 (a) shows the time-averaged streamwise velocity on the centerline direction of the present simulations. They are in accordance with the experiments of Lourenco and Shih (data from Beaudan and Moin [2]) and Ong and Wallace [23], even this is probably a sign of lack of grid resolution, as discussed in [15]. We can notice an improvment of VMS-LES over classical LES method.

Figure 2 (b) shows the pressure distribution on the cylinder surface averaged in time and on homogeneous z direction. The results from the simulations are very close each other on the whole cylinder and it appears a deviation with the



(a) (b)

Fig. 2. (a) Time averaged and z-averaged streamwise velocity on the centerline direction, experiments: Lourenco and Shih [19] (LS) and Ong and Wallace [23] (OW) (b) Time averaged and z-averaged pressure distribution on the surface of the cylinder, experiments: Norberg [22]

experimental data from Norberg [22]. The discrepancies visible between 60 and 100 degrees may be explained by the rather coarse grid used.

The Fourier energy spectrum of the spanwise velocity at $\mathrm{P}(3, 0.5, 0)$ for Vreman SGS model with LES and VMS-LES methodology is displayed in Figure 3. The frequency is nondimensionalized by the Strouhal shedding frequency. Via the Taylor hypothesis of frozen turbulence (which is justified since the mean convection velocity is large at that point) which allows to assume that high (low) time frequencies correspond to small (large) scale in space, we observe that the energy in the large resolved scales are higher with VMS-LES than with LES. These results corroborate the fact that in the VMS-LES approach, the modeling of the energy dissipation effects of the unresolved scales affects only the small resolved scales contrary to the LES approach in which these dissipative effects act on all the resolved scales.

For this problem involving 1.5 million degrees of freedom and for twenty shedding cycles simulation, the simulation time is about 7 hours on a 32-processor IBM Power 4 computer.

5 Hybrid RANS/VMS-LES Simulations

The new proposed hybrid model (*Fluctuation Correction Model*,FCM) has been applied to the simulation of the flow around a circular cylinder at $Re = 140000$ (based on the far-field velocity and the cylinder diameter). The domain dimensions are: $-5 \leq x/D \leq 15$, $-7 \leq y/D \leq 7$ and $0 \leq z/D \leq 2$ (the symbols are the same as in Section 4). Two grids have been used, the first one (GR1) has $4.6{\times}10^5$ nodes, while the second one has (GR2) $1.4{\times}10^6$ nodes. Both grids are composed of a structured part around the cylinder boundary and a unstructured part in the rest of the domain. The inflow conditions are the same as in the DES simulations of Travin et al. [30]. In particular, the flow is assumed

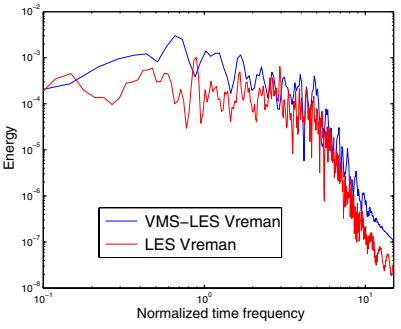


Fig. 3. Fourier energy spectrum: spanwise velocity for LES Vreman and VMS-LES Vreman

Table 2. Simulation name and their main characteristics

Simulation	Blending parameter	Grid	LES-SGS model
FCM1	VR	GR1	Smagorinsky
FCM2	LR	GR1	Smagorinsky
FCM3	LR	GR2	Smagorinsky
FCM4	LR	GR1	Vreman
FCM5	LR	GR1	Wale

to be highly turbulent by setting the inflow value of eddy-viscosity to about 5 times the molecular viscosity as in the DES simulation of Travin et al. [30]. This setting corresponds to a free-stream turbulence level $\overline{u'^2}/U_0$ (where u' is the inlet velocity fluctuation and U_0 is the free-stream mean velocity) of the order of 4%. As discussed also by Travin et al. [30], the effect of such a high level of free-stream turbulence is to make the boundary layer almost entirely turbulent also at the relatively moderate considered Reynolds number. The boundary treatment is the same as for simulations using VMS-LES approach in Section 4, except that wall laws are now used. The RANS model is that based on the low-Reynolds approach [8]. The LES closure is based on the VMS approach (see Section2.1). The SGS models used in the simulations are those given in Section2. The main parameters characterizing the simulations carried out with the FCM are summarized in Table 2.

The main flow bulk parameters obtained in the present simulations are summarized in Table 3, together with the results of DES simulations in the literature and some experimental data. They have been computed by averaging in time, over at least 17 shedding cycles and in the spanwise direction. Let us analyze, first, the sensitivity to the blending parameter, by comparing the results of the simulation FCM1 and FCM2. The results are practically insensitive to the definition of the blending parameter. Conversely, the grid refinement produces a delay in the boundary layer separation which results in a decrease of $\bar{C}_d$ (compare FCM2 and FCM3). However, note that, for unstructured grids, the refinement changes the local quality of the grid (in terms of homogeneity and regularity of

Table 3. Main bulk flow quantities for the circular cylinder test case. Same notations as in Table 1, θ_{sep} the separation angle.

Data from	Re	$\overline{C_d}$	C_l'	St	l_r	θ_{sep}
FCM1	**1.4 10^5**	**0.62**	**0.083**	**0.30**	**1.20**	**108**
FCM2	**1.4 10^5**	**0.60**	**0.082**	**0.31**	**1.15**	**113**
FCM3	**1.4 10^5**	**0.54**	**0.065**	**0.33**	**1.13**	**115**
FCM4	**1.4 10^5**	**0.62**	**0.127**	**0.28**	**1.19**	**117**
FCM5	**1.4 10^5**	**0.60**	**0.083**	**0.30**	**1.23**	**114**
Numerical data						
DES [30]	1.4 10^5	0.57-0.65	0.08-0.1	0.28-0.31	1.1 -1.4	93-99
DES [18]	1.4 10^5	0.6-0.81	–	0.29-0.3	0.6-0.81	101-105
Experiments						
[11]	3.8 10^6	0.58	–	0.25	–	110
[1]	5 10^6	0.7	–	–	–	112
[24]	8 10^6	0.52	0.06	0.28	–	–

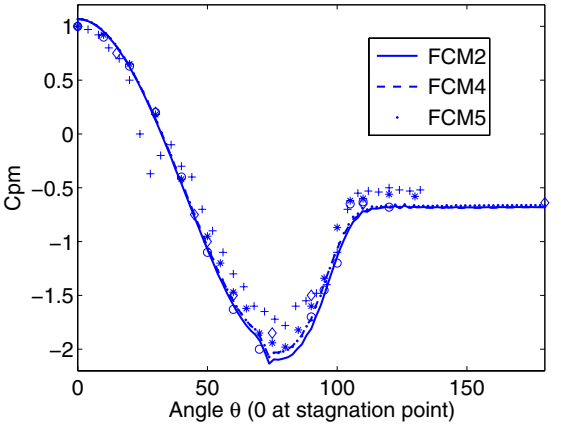


Fig. 4. $\bar{C}_p$ on the cylinder surface compared to numerical and experimental results: $*$ Jones, $+$ James, o Roshko, $\diamond$ Travin et al

the elements) and this may enhance the sensitivity of the results. The sensitivity to the VMS-LES closure model is also quiet low (compare FCM2, FCM4 and FCM5). This low sensitivity has been observed also in VMS-LES simulations at low Reynolds number see Section4 and, thus, it seems more peculiar to the VMS-LES approach rather than to the hybrid model.

The agreement with the DES results is fairly good. As for the comparison with the experiments, as also stated in Travin et al. [30], since our simulations are characterized by a high level of turbulence intensity at the inflow, it makes sense to compare the results with experiments at higher Reynolds number, in which, although the level of turbulence intensity of the incoming flow is very low, the transition to turbulence of the boundary layer occurs upstream separation.

The agreement with these high Re experiments is indeed fairly good, as shown in Table 3 and in Figure 4. The behavior of the separation angle requires a brief discussion. There is a significant discrepancy between the values obtained in DES and the experimental ones. For our simulations, the values of θ_{sep} shown in Table 3 are estimated by considering the point at which the C_p distribution over the cylinder becomes nearly constant (see e.g. Figure 4), as usually done in experimental studies. Indeed, the reported values are generally in better agreement with the experiments than those obtained by DES. Finally, the model works in RANS mode in the boundary layer and in the shear-layers detaching from the cylinder, while in the wake a full VMS-LES correction is recovered.

For this problem involving 3.2 million degrees of freedom and for twenty shedding cycles simulation, the simulation time is about 30 hours on a 32-processor IBM Power 4 computer and about 16 hours on a 32-processor IBM Power 5 computer.

6 Conclusion

In this paper we have presented parallel simulations of three-dimensional turbulent flows. We have first investigated the application of a Variational multiscale LES for the simulations of a flow past a circular cylinder at a subcritical Reynolds number equal to Re =3900. Although a rather coarse grid has been used, this model gives accurate predictions of bulk coefficients and shows that two recently developed SGS models, the Vreman's model and the WALE model combine well in the VMS formulation. Moreover, it appears in this approach that the influence of the SGS model is weak, but this seems to give a support to the VMS idea of adding some dissipation only to the smallest resolved scales. In a second part, we have presented a hybrid RANS/LES approach using different definitions of blending parameter and SGS models. For the closure of the LES part, the VMS approach has been used. This model is validated on the prediction of a flow around a circular cylinder at higher supercritical Reynolds number (Re =140000). The results obtained correlate well with the experimental and numerical data from the literature as well as the behavior of the blending function.

Acknowledgments. CINES (Montpellier, France) and CINECA (Bologna, Italy) are gratefully acknowledged for having provided the computational resources.

References

1. Achenbach, E.: Distribution of local pressure and skin friction around a circular cylinder in cross-flow up to Re= 5×10^6. J. Fluid Mech. 34(4), 625–639 (1968)
2. Beaudan, P., Moin, P.: Numerical experiments on the flow past a circular cylinder at sub-critical reynolds number. Report No. TF-62 (1994)
3. Breuer, M.: Numerical and modeling on large eddy simulation for theflow past a circular cylinder. International Journal of Heat and Fluid Flow 19, 512–521 (1998)

4. Camarri, S., Salvetti, M.V., Koobus, B., Dervieux, A.: A low diffusion MUSCL scheme for LES on unstructured grids. Computers and Fluids 33, 1101–1129 (2004)
5. Cardell, G.S.: Flow past a circular cylinder with a permeable splitter plate. Ph.D Thesis (1993)
6. Collis, S.S.: The dg/vms method for unified turbulence simulation. AIAA Paper (2002)
7. Pagano, G., Camarri, S., Salvetti, M.V., Koobus, B., Dervieux, A.: Strategies for rans/vms-les coupling. Technical Report RR-5954, INRIA (2006)
8. Goldberg, U., Peroomian, O., Chakravarthy, S.: A wall-distance-free $k - \varepsilon$ model with enhanced near-wall treatment. Journal of Fluids Engineering 120, 457–462 (1998)
9. Hughes, T.J.R., Mazzei, L., Jansen, K.E.: Large eddy simulation and the variational multiscale method. Comput. Vis. Sci. 3, 47–59 (2000)
10. Lee, S., Lee, J., Park, N., Choi, H.: A dynamical subgrid-scale eddy viscosity model with a global model coefficient. Physics of Fluids (2006)
11. James, W.D., Paris, S.W., Malcolm, G.V.: Study of viscous cross flow effects on circular cylinders at high Reynolds numbers. AIAA Journal 18, 1066–1072 (1980)
12. Sarkis, M., Koobus, B.: A scaled and minimum overlap restricted additive Schwarz method with application to aerodynamics. Comput. Methods Appl. Mech. Eng. 184, 391–400 (2000)
13. Koobus, B., Farhat, C.: A variational multiscale method for the large eddy simulation of compressible turbulent flows on unstructured meshes-application to vortex shedding. Comput. Methods Appl. Mech. Eng. 193, 1367–1383 (2004)
14. Koobus, B., Camarri, S., Salvetti, M.V., Wornom, S., Dervieux, A.: Parallel simulation of three-dimensional complex flows: Application to two-phase compressible flows and turbulent wakes. Adv. Eng. Software 38, 328–337 (2007)
15. Kravchenko, A.G., Moin, P.: Numerical studies of flow over a circular cylinder at $re_d = 3900$. Physics of fluids 12, 403–417 (1999)
16. Labourasse, E., Sagaut, P.: Reconstruction of turbulent fluctuations using a hybrid RANS/LES approach. J. Comp. Phys. 182, 301–336 (2002)
17. Lanteri, S.: Parallel Solutions of Three Dimensional Compressible Flows. Technical Report RR-2594, INRIA (1995)
18. Lo, S.-C., Hofmann, K.A., Dietiker, J.-F.: Numerical investigation of high Reynolds number flows over square and circular cylinder. Journal of Thermophysics and Heat Transfer 19, 72–80 (2005)
19. Lourenco, L.M., Shih, C.: Characteristics of the plane turbulent near weak of a circular cylinder. a particle image velocimetry study
20. Salvetti, M.V., Koobus, B., Camarri, S., Dervieux, A.: Simulation of bluff-body flows thorugh a hybrid rans/vms-les model. In: Proceedings of the IUTAM Symposium on Unsteady Separated Flows and their Control, Corfu (Grece), June 18-22 (2007)
21. Nicoud, F., Ducros, F.: Subgrid-scale stress modelling based on the square of the velocity gradient tensor. Flow, Turbulence and Combustion 62, 183–200 (1999)
22. Norberg, C.: Effects of reynolds number and low-intensity free-sream turbulence on the flow around a circular cylinder. Publ. No. 87/2, Department of Applied Termosc. and Fluid Mech (1987)
23. Ong, L., Wallace, J.: The velocity field of the turbulent very near wake of a circular cylinder. Exp. in Fluids 20, 441–453 (1996)
24. Schewe, J.W.: On the forces acting on a circular cylinder in cross flow from subcritical up to transcritical Reynolds numbers. J. Fluid Mech. 133, 265–285 (1983)

25. Smagorinsky, J.: General circulation experiments with the primitive equations. Monthly Weather Review 91(3), 99–164 (1963)
26. Son, J., Hanratt, T.J.: Velocity gradient of the wall for flow around a circular cylinder at reynolds nimbers from 5×10^3 to 10^5. J. Fluid Mech. 35, 353–368 (1969)
27. Spalart, P.R., Jou, W.H., Strelets, M., Allmaras, S.: Advances in DNS/LES. In: Comments on the feasibility of LES for wings and on a hybrid RANS/LES approach, Columbus, OH (1997)
28. Steger, J.L., Warming, R.F.: Flux vector splitting for the inviscid gas dynamic equations with applications to the finite difference methods. J. Comp. Phys. 40(2), 263–293 (1981)
29. Mazzei, L., Hughes, T.J.R., Oberai, A.A.: Large eddy simulation of turbulent channel flows by the variational multiscale method. Phys. Fluids 13, 1784–1799 (2001)
30. Travin, A., Shur, M., Strelets, M., Spalart, P.: Detached-eddy simulations past a circular cylinder. Flow, Turbulence and Combustion 63, 293–313 (1999)
31. Vreman, A.W.: An eddy-viscosity subgrid-scale model for turbulent shear flow: algebraic theory and application. Physics of Fluids 16, 3670–3681 (2004)

Vortex Methods for Massively Parallel Computer Architectures

Philippe Chatelain[1], Alessandro Curioni[2], Michael Bergdorf[1], Diego Rossinelli[1], Wanda Andreoni[2], and Petros Koumoutsakos[1]

[1] Computational Science and Engineering Laboratory
ETH Zurich, CH-8092, Switzerland
Tel.: +41 44 632 7159
Fax: +41 44 632 1703
[2] Computational Sciences
IBM Research Division - Zurich Research Laboratory
Saumerstrasse 4, CH-8003 Rueschlikon, Switzerland
Tel.: +41 44 724 8633
Fax: +41 44 724 8958

Abstract. We present Vortex Methods implemented in massively parallel computer architectures for the Direct Numerical Simulations of high Reynolds numbers flows. Periodic and non-periodic domains are considered leading to unprecedented simulations using billions of particles. We discuss the implementation performance of the method up to $16k$ IBM BG/L nodes and the evolutionary optimization of long wavelength instabilities in aircraft wakes.

1 Introduction

Vortex methods exemplify the computational advantages and challenges of particle methods in simulations of incompressible vortical flows. These simulations are based on the discretization of the vorticity-velocity formulation of the Navier-Stokes equations in a Lagrangian form.

In the recent years hybrid techniques (see [1,2] and references therein) have been proposed where a mesh is used along with the particles in order to develop efficient and accurate computations of vortical flows.

In this work, we present an efficient and scalable implementation of these methodological advances for the massively parallel architecture of the IBM BG/L. The present results involve DNS on $4k$ processors and an efficiency investigation going up to $16k$ processors and 6 billion particles.

The method is applied to the decay of aircraft wakes and vortex rings. The wake of an aircraft consists of long trailing vortices that can subject the following aircraft to a large downwash. Several research efforts have focused on the identification of the governing physical mechanisms of wake evolution that would lead to design of vortex wake alleviation schemes[3,4,5,6,7]. Flight realistic conditions involve turbulent flows ($Re \sim 10^6$) in unbounded domains for which DNS reference data is still lacking.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 479–489, 2008.

State of the art simulations have been limited to low resolution LES in large domains[8], or vortex method simulations[9,10] which achieved Re=5000 DNS in short domains and investigated various subgrid stress models for LES in long domains.

The present work enables unprecedented resolutions for the DNS of long wavelength instabilities. The long domain calculation at Re=6000 presented herein constitutes the largest DNS ever achieved for a vortex particle method. We also present results for the turbulent decay of a vortex ring at $Re_\Gamma = 7500$. Ongoing work includes simulations at even higher Reynolds on larger partitions of BG/L, the development of unbounded conditions and the coupling of this methodology with evolutionary algorithms in order to accelerate the decay and mixing inside these vortical flows.

2 Methodology

2.1 The Remeshed Vortex Particle Method

We consider a three dimensional incompressible flow and the Navier-Stokes equations in its velocity ($\mathbf{u}$)-vorticity ($\boldsymbol{\omega} = \nabla \times \mathbf{u}$) form :

$$\frac{D\boldsymbol{\omega}}{Dt} = (\boldsymbol{\omega} \cdot \nabla)\,\mathbf{u} + \nu \nabla^2 \boldsymbol{\omega} \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2}$$

where $\frac{D}{Dt} = \frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla$ denotes the Lagrangian derivative and ν is the kinematic viscosity.

Vortex methods discretize the vorticity field with particles, characterized by a position $\mathbf{x}_p$, a volume V_p and a strength $\boldsymbol{\alpha}_p = \int_{V_p} \boldsymbol{\omega} d\mathbf{x}$. The field is then

$$\boldsymbol{\omega}(\boldsymbol{x}, t) \approx \sum_p \boldsymbol{\alpha}_p(t) \zeta^h \left(\mathbf{x} - \mathbf{x}_p(t)\right) , \tag{3}$$

where ζ is the interpolation kernel and h the mesh spacing. Particles are convected by the flow field and their strength undergoes vortex stretching and diffusion

$$\begin{aligned} \frac{d\mathbf{x}_p}{dt} &= \mathbf{u}(\mathbf{x}_p)\,, \\ \frac{d\boldsymbol{\alpha}_p}{dt} &= \int_{V_p} (\boldsymbol{\omega} \cdot \nabla)\mathbf{u} + \nu \nabla^2 \boldsymbol{\omega} d\mathbf{x}\,, \\ &\simeq \left((\boldsymbol{\omega} \cdot \nabla)\mathbf{u}(\mathbf{x}_p) + \nu \nabla^2 \boldsymbol{\omega}(\mathbf{x}_p)\right) V_p\,. \end{aligned} \tag{4}$$

Using the definition of vorticity and the incompressibility constraint the velocity field is computed by solving the Poisson equation

$$\nabla^2 \mathbf{u} = -\nabla \times \boldsymbol{\omega}\,. \tag{5}$$

The solution of this equation can be computed by using the Green's function solution of the Poisson equation or, as in the present hybrid formulation, grid solvers.

The use of a mesh (M) conjointly with the particles (P) allows the use of efficient tools such as grid solvers and Finite Differences. This is demonstrated below in the case of a Euler time-step

– (P → M) Interpolate particle strengths on a lattice by evaluating Eq. 3 on grid locations

$$\boldsymbol{\omega}(\mathbf{x}_{ij\ldots}) = \sum_p \boldsymbol{\alpha}_p \zeta^h \left(\mathbf{x}_{ij\ldots} - \mathbf{x}_p\right) \tag{6}$$

where $\mathbf{x}_{ij\ldots}$ is a grid node and $ij\ldots$ are node indices.

– (M → M) Perform operations on the grid, *i.e.* solve the Poisson equation for velocity in Fourier space, use Finite Differences and evaluate the right-hand sides of the system of Eq. 4

– (M → P) Interpolate velocities, right-hand sides, respectively back onto the particles,

$$\begin{aligned} \mathbf{u}(\mathbf{x}_p) &= \sum_i \sum_j \sum_{\ldots} h^{-d} \mathbf{u}(\mathbf{x}_{ij\ldots}) \zeta^h \left(\mathbf{x}_p - \mathbf{x}_{ij\ldots}\right) \\ \frac{D\boldsymbol{\omega}}{Dt}(\mathbf{x}_p) &= \sum_i \sum_j \sum_{\ldots} h^{-d} \frac{D\boldsymbol{\omega}}{Dt}(\mathbf{x}_{ij\ldots}) \zeta^h \left(\mathbf{x}_p - \mathbf{x}_{ij\ldots}\right) \end{aligned} \tag{7}$$

and advance the quantities and locations.

The Lagrangian distortion of the particles leads to loss of convergence[11,12]. We ensure accuracy by means of a periodic reinitialization of the particle locations [13,14,15,16,1]. This *remeshing* procedure, essentially a P → M interpolation, is performed at the end of every time step and uses the third order accurate M_4' kernel[17].

2.2 Implementation for Parallel Computer Architectures

The method was implemented as a client application of the open source Parallel Particle Mesh (PPM) library[18]. PPM provides a general-purpose framework that can handle the simulation of particle-only, mesh-only or particle-mesh systems. The library defines topologies, i.e. space decompositions and the assignment of sub-domains to processors, which achieve particle- and mesh-based load balancing. The library provides several tools for the efficient parallelization of the particle-mesh approach described in Section 2.1. Data communication is organized in *local* and *global* mappings. Local mappings handle

– the advection of particles from a sub-domain into another
– ghost mesh points for the consistent summation of particle contributions along sub-domain boundaries, e.g. in the P → M step: the interpolation stencil will distribute particle strength to ghost points outside its own sub-domain
– ghost mesh points for consistent Finite Difference operations.

Global mappings are used for the transfer of mesh data from a topology to another, as in the case of the pencil topologies involved in multi-dimensional FFTs. PPM is written in Fortran 90 on top of the Message Passing Interface (MPI); the client uses the FFTW library[19] inside the Fourier solver.

The code is run on an IBM Blue Gene/L solution with dual cores nodes based on the PowerPC 440 700Mhz low power processor. Each node has 512MB of memory. The computations are all carried out in co-processor mode: one of the two CPUs is fully devoted to the communications. The machine used for production was the BG/L at IBM T.J. Watson Research Center - Yorktown Heights[1] whereas porting, optimization and testing was done on the BG/L system of the IBM Zurich Research Laboratory. Machine dependent optimization consisted in

1. data reordering and compiler directives to exploit the double floating point unit of the PowerPC 440 processors,
2. mapping of the cartesian communicators to the BG/L torus,
3. use of the BG/L tree network for global reductions.

3 Aircraft Wakes

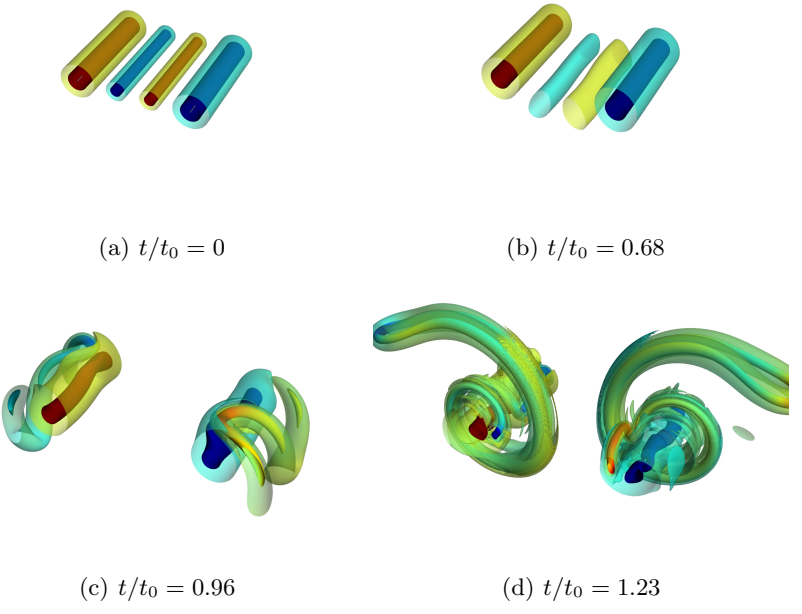
The evolution and eventual destruction of the trailing vortices is affected by several types of instabilities, usually classified according to their wavelength. Long wavelength instabilities are the most powerful to drive the collapse of a vortex pair albeit with a slow growth rate. The well-known Crow instability[20] is an example of such instabilities that deforms the vortex lines into sinusoidal structures until vortices of opposite sign reconnect and form rings.

More complex systems with multiple vortex pairs can undergo other instabilities. A rapidly growing, medium-wavelength instability has been the focus of recent experimental [5,7,21] and numerical studies[8,9,10]. This instability occurs in the presence of a secondary vortex pair that is counter-rotating relative to the main pair. These secondary vortices are generated by a sufficient negative load on the horizontal tail or the inboard edge of outboard flaps. Being weaker, they eventually wrap around the primary ones in so-called Ω-loops, leading to the reconnection of vortices of unequal circulations. This in turn triggers an accelerated vortex destruction.

3.1 Convergence and Scalability

We use the geometry of this particular medium wavelength instability to assess the performance of our code. The geometry of the problem is taken from [9]; it comprises two counter-rotating vortex pairs with spans b_1, b_2 and circulations Γ_1, Γ_2. The Reynolds number is $Re = \Gamma_0/\nu = 3500$, where $\Gamma_0 = \Gamma_1 + \Gamma_2$. Three grid sizes were considered, $64 \times 320 \times 192$, $128 \times 640 \times 384$, and $256 \times 1280 \times 768$, resulting in 4, 32 and 252 million particles respectively. All three configurations were run on 1024 processors of IBM BG/L. The time-step was kept

[1] Compiled with XLF version 10.1, with BG/L driver V1.3 and FFTW 3.1.1.



(a) $t/t_0 = 0$ (b) $t/t_0 = 0.68$

(c) $t/t_0 = 0.96$ (d) $t/t_0 = 1.23$

Fig. 1. Medium-wavelength instability of counter-rotating vortices, 128×640×384-grid: evolution of vorticity iso-surfaces. The opaque surface corresponds to $|\boldsymbol{\omega}| = 10\Gamma_1/b_1^2$; the transparent one, to $|\boldsymbol{\omega}| = 2\Gamma_1/b_1^2$.

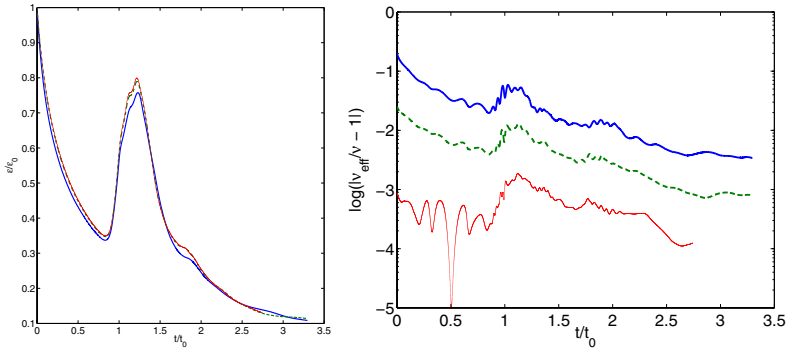
constant for all resolutions $\Delta t = 3.3\,10^{-4} t_0$ where $t_0 = \frac{2\pi b_0^2}{\Gamma_0}$ and $b_0 = \frac{\Gamma_1 b_1 + \Gamma_2 b_2}{\Gamma_0}$. Figure 1 shows the evolution of vorticity iso-surfaces and the wrapping-around of the secondary vortices around the main ones. Diagnostics (Fig. 2) such as the evolution of enstrophy, which measures the energy decay and the evolution of the effective numerical viscosity confirm the low dissipation of the method and its convergence.

The parallel scalability was assessed for $512 \leq N_{\text{CPU}} \leq 16384$ on IBM BG/L. We measure the strong efficiency as

$$\eta_{\text{strong}} = \frac{N_{\text{CPUS}}^{\text{ref}}\, T(N_{\text{CPUS}}^{\text{ref}})}{N_{\text{CPUS}}\, T(N_{\text{CPUS}})} \tag{8}$$

where T is the average computation time of one time step. In order to test the code up to the large sizes allowed by BG/L, we used $N_{\text{CPUS}}^{\text{ref}} = 2048$ and a problem size of $768 \times 1024 \times 2048$ or 1.6 billion particles. This brings the per-processor problem size from 786432 down to 98304 when we run on the maximum number of processors. The curve (Fig. 3(b)) displays a plateau up to $N_{\text{CPUS}} = 4096$, with the per-processor problem size becoming progressively smaller and communication overhead overwhelming the computing cycles.

From this result, we base our weak scalability study on a constant per-processor number of particles of $M_{\text{per CPU}} \simeq 4\,10^5$. We used the following



(a) Enstrophy, $\epsilon = \int \boldsymbol{\omega} \cdot \boldsymbol{\omega}\, dV$ (b) Relative error in the effective kinematic viscosity, $\nu_{\text{effective}} = -\frac{dE/dt}{\epsilon}$

Fig. 2. Medium-wavelength instability of counter-rotating vortices: convergence and diagnostics for three spatial resolutions: $N_x = 64$ (solid thick), 128 (dashed) and 256 (solid thin)

measure

$$\eta_{\text{weak}} = \frac{T(N^{\text{ref}}_{\text{CPUS}}, M^{\text{ref}})}{T(N_{\text{CPUS}}, \frac{N_{\text{CPUS}}}{N^{\text{ref}}_{\text{CPUS}}} M^{\text{ref}})} \,. \tag{9}$$

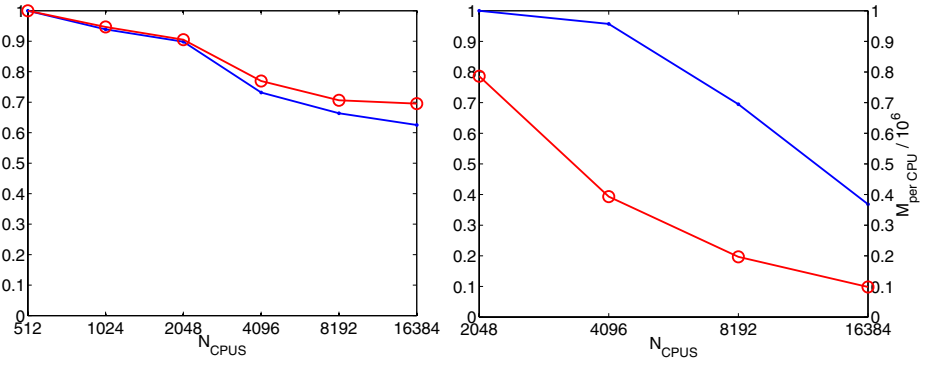
where we took $N^{\text{ref}}_{\text{CPUS}} = 512$. The code displays (Fig. 3(a)) excellent scalability up to $N_{\text{CPUS}} = 4096$. Eq. 9 assumes linear complexity for the problem at hand. There is however an $O(N \log N)$ component to the overall complexity of the present problem as we are solving the Poisson equation for the convection velocity. The two curves (with and without the cost for the solution of the Poisson equation) are shown in (Fig. 3(a)); the relatively small gap between the two curves manifests the good performance of the Poisson solver.

3.2 Instability Initiation by Ambient Noise in a Large Domain

We consider the configuration presented in the state of the art calculations in [8, see configuration 2] simulating the onset of instabilities of multiple wavelengths in a long domain. The domain length is chosen as the wavelength of maximum growth rate for the Crow instability, $L_x = 9.4285 b_1$. The transversal dimensions are $L_y = 1/2\, L_x$ and $L_z = 3/8\, L_x$. The vortices have Gaussian cores

$$\omega(r) = \frac{1}{2\pi\sigma^2} \exp(-(r/2\sigma)^2) \tag{10}$$

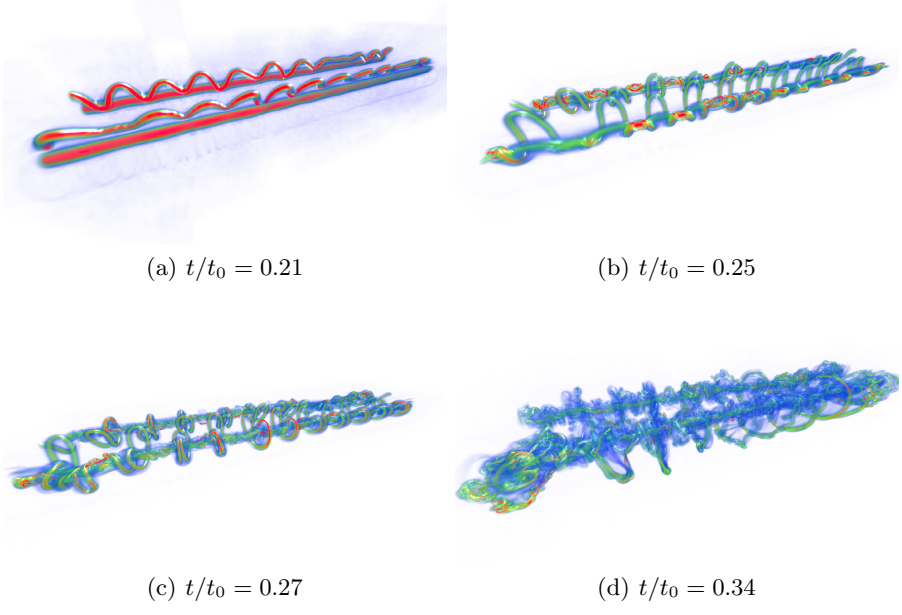
with $\sigma_1/b_1 = 0.05$ and $\sigma_2/b_1 = 0.025$. The secondary pair is located at $b_2/b_1 = 0.5$, with a relative strength $\Gamma_2/\Gamma_1 = -0.35$. In addition to the initially



(a) Weak scalability for a per-processor problem size = $4\,10^5$;full problem (solid dots) and excluding the Poisson solver (circles)

(b) Strong scalability (solid dots) and per-processor size (circles)

Fig. 3. Medium-wavelength instability of counter-rotating vortices: parallel efficiencies on IBM BlueGene/L



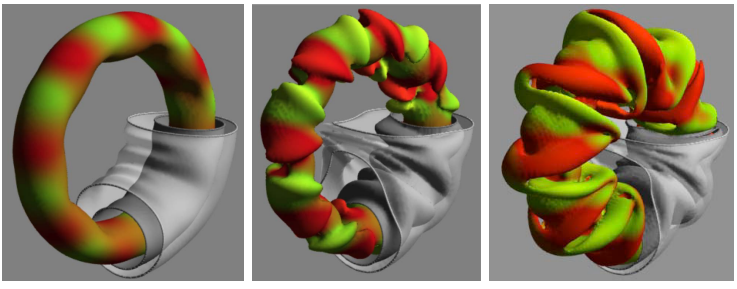
(a) $t/t_0 = 0.21$

(b) $t/t_0 = 0.25$

(c) $t/t_0 = 0.27$

(d) $t/t_0 = 0.34$

Fig. 4. Counter-rotating vortices in a periodic domain, initiation by ambient noise: visualization of the vorticity structures by volume rendering. High vorticity norm regions correspond to red and opaque; low vorticity are blue and transparent.



(a) (b) (c)

Fig. 5. Evolution of a Vortex ring at Re=7500: vorticity iso-surfaces colored by the stream-wise component of vorticity

unperturbed vortices, the vorticity field is filled with a white noise that produces $u_{\mathrm{RMS}} = 0.005\, u_{\max}$. We study this flow with DNS at $Re_{\Gamma_1} = 6000$. This represents a three-fold increase over previously reported Reynolds numbers [8]. In addition, these prior simulations used a coarse resolution and a crude LES model (MILES[22]) to model the high Reynolds number dynamics of the flow. The present DNS is afforded due to a mesh resolution of $2048 \times 1024 \times 768$ and 1.6 billion particles. It is run on 4096 CPUs; the wall-clock computation time was 39s on average per time step. With approximately 10000 time steps, this represents a time-to-solution of 100 hours.

Figure 4 shows that this system with a random initial condition picks up the medium-wavelength instability. At $t/t_0 = 0.25$ (Fig. 4(b)), we count 10 and 11 Ω-loops along the two primary vortices. This corresponds to the average wavelengths $\lambda/b_1 = 0.943$ and 0.86. These values are sensibly different from the ones reported in [8], 1.047 and 1.309. This comparison, however, considers the problem at the end of the exponential growth and ignores the uneven distribution of loop wavelengths and hence, individual growth rates.

4 Vortex Rings

The same code has been applied to the turbulent decay of vortex rings at $Re_\Gamma = 7500$[23]. It allowed the analysis of the vortex dynamics in the non-linear stage and their correlation with structures captured in dye visualization and an observed decay of circulation. Figure 5 shows the emergence of stream-wise structures in the ring.

5 Extensions

5.1 Unbounded Poisson Solvers

As mentioned in Section 2.2, the Poisson equation for velocity (Eq. 5) is solved on a grid in Fourier space. This approach exploits the good scalability of Fourier

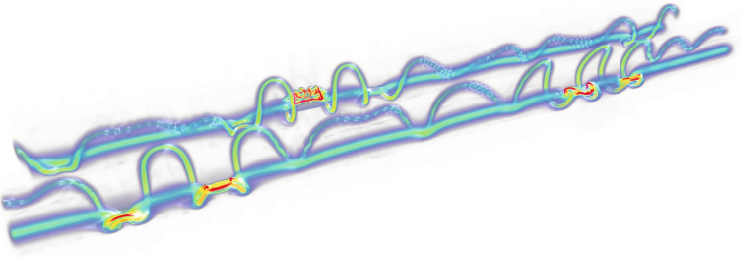


Fig. 6. Counter-rotating vortices in an unbounded domain, initiation by ambient noise: vorticity structures at $t/t_0 = 0.22$

transforms and the associated data mappings but it imposes the use of large domains in order to mitigate the effect of the periodic images. We have implemented solvers which carry out the fast convolutions of the unbounded Green's functions in Fourier space[24]. As a result, the same parallel FFTs can be used and even combined to achieve mixed periodic-unbounded conditions. We note that this can also be carried out with Fast Multipole Methods[9,10] but at the cost of more complex communication patterns. Figure 6 shows results for the medium wavelength instability in an unbounded domain at $Re_{\Gamma_1} = 8000$. The higher Reynolds number is afforded thanks to the unbounded solver computational savings; it allows short wavelength instabilities to develop inside the vortex cores. An in-depth analysis of this method is currently under preparation[25].

5.2 Wake Optimization

Our vortex code has been coupled to an Evolution Strategy(ES), here with Covariance Matrix Adaptation[26], in order to accelerate the decay of a wake. The wake consists of perturbed co-rotating pairs[3]; this model approximates wing tip and flap vortices and the effect of an active device. The ES searches the space of the parameters describing the wake base structure and the perturbation. The performance of each configuration is measured by a scalar objective function, e.g. energy decay. Each function evaluation thus entails the computation of a transient flow on large partitions of parallel computers (128 to 512 CPUs for approximately 10 wallclock hours).

6 Conclusions

This paper presents the implementation of an efficient particle-mesh method for massively parallel architectures and its application to wakes. We refer to [27] for a more extensive assessment of the method.

Our code displays good scalability up to $16K$ processors on BlueGene/L. The origin of the parallel efficiency drop at $4K$ is being investigated; a possible cause

is the recurrent computation of mesh intersections inside the global mappings. Other code development efforts include the implementation of unbounded and non-periodic boundary conditions. Finally, the optimization of vortex dynamics for enhanced decay and mixing is the subject of ongoing investigations.

References

1. Winckelmans, G.: Vortex methods. In: Stein, E., De Borst, R., Hughes, T.J. (eds.) Encyclopedia of Computational Mechanics, vol. 3. John Wiley and Sons, Chichester (2004)
2. Koumoutsakos, P.: Multiscale flow simulations using particles. Annu. Rev. Fluid Mech. 37, 457–487 (2005)
3. Crouch, J.D.: Instability and transient growth for two trailing-vortex pairs. Journal of Fluid Mechanics 350, 311–330 (1997)
4. Crouch, J.D., Miller, G.D., Spalart, P.R.: Active-control system for breakup of airplane trailing vortices. AIAA Journal 39(12), 2374–2381 (2001)
5. Durston, D.A., Walker, S.M., Driver, D.M., Smith, S.C., Savas, Ö.: Wake vortex alleviation flow field studies. J. Aircraft 42(4), 894–907 (2005)
6. Graham, W.R., Park, S.W., Nickels, T.B.: Trailing vortices from a wing with a notched lift distribution. AIAA Journal 41(9), 1835–1838 (2003)
7. Ortega, J.M., Savas, Ö.: Rapidly growing instability mode in trailing multiple-vortex wakes. AIAA Journal 39(4), 750–754 (2001)
8. Stumpf, E.: Study of four-vortex aircraft wakes and layout of corresponding aircraft configurations. J. Aircraft 42(3), 722–730 (2005)
9. Winckelmans, G., Cocle, R., Dufresne, L., Capart, R.: Vortex methods and their application to trailing wake vortex simulations. C. R. Phys. 6(4-5), 467–486 (2005)
10. Cocle, R., Dufresne, L., Winckelmans, G.: Investigation of multiscale subgrid models for les of instabilities and turbulence in wake vortex systems. Lecture Notes in Computational Science and Engineering 56 (2007)
11. Beale, J.T., Majda, A.: Vortex methods I: convergence in 3 dimensions. Mathematics of Computation 39(159), 1–27 (1982)
12. Beale, J.T.: On the accuracy of vortex methods at large times. In: Proc. Workshop on Comput. Fluid Dyn. and React. Gas Flows, IMA, Univ. of Minnesota, 1986, p. 19. Springer, New York (1988)
13. Cottet, G.H.: Artificial viscosity models for vortex and particle methods. J. Comput. Phys. 127(2), 299–308 (1996)
14. Koumoutsakos, P.: Inviscid axisymmetrization of an elliptical vortex. J. Comput. Phys. 138(2), 821–857 (1997)
15. Chaniotis, A., Poulikakos, D., Koumoutsakos, P.: Remeshed smoothed particle hydrodynamics for the simulation of viscous and heat conducting flows. J. Comput. Phys. 182, 67–90 (2002)
16. Eldredge, J.D., Colonius, T., Leonard, A.: A vortex particle method for two dimensional compressible flow. J. Comput. Phys. 179, 371–399 (2002)
17. Monaghan, J.J.: Extrapolating b splines for interpolation. J. Comput. Phys. 60(2), 253–262 (1985)
18. Sbalzarini, I.F., Walther, J.H., Bergdorf, M., Hieber, S.E., Kotsalis, E.M., Koumoutsakos, P.: PPM a highly efficient parallel particle mesh library for the simulation of continuum systems. J. Comput. Phys. 215, 566–588 (2006)

19. Frigo, M., Johnson, S.G.: FFTW: An adaptive software architecture for the FFT. In: Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing, Seattle, WA, vol. 3, pp. 1381–1384 (1998)
20. Crow, S.C.: Stability theory for a pair of trailing vortices. AIAA Journal 8(12), 2172–2179 (1970)
21. Stuff, R.: The near-far relationship of vortices shed from transport aircraft. In: AIAA (ed.) AIAA Applied Aerodynamics Conference, 19th, Anaheim, CA, AIAA, pp. 2001–2429. AIAA, Anaheim (2001)
22. Boris, J.P., Grinstein, F.F., Oran, E.S., Kolbe, R.L.: New insights into large eddy simulation. Fluid Dynamics Research 10(4-6), 199–228 (1992)
23. Bergdorf, M., Koumoutsakos, P., Leonard, A.: Direct numerical simulations of vortex rings at $re_\gamma = 7500$. J. Fluid. Mech. 581, 495–505 (2007)
24. Hockney, R., Eastwood, J.: Computer Simulation using Particles. Institute of Physics Publishing (1988)
25. Chatelain, P., Koumoutsakos, P.: Fast unbounded domain vortex methods using fourier solvers (in preparation, 2008)
26. Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evol. Comput. 11(1), 1–18 (2003)
27. Chatelain, P., Curioni, A., Bergdorf, M., Rossinelli, D., Andreoni, W., Koumoutsakos, P.: Billion vortex particle direct numerical simulations of aircraft wakes. Computer Methods in Applied Mechanics and Engineering 197(13), 1296–1304 (2008)

On the Implementation of Boundary Element Engineering Codes on the Cell Broadband Engine

Manoel T.F. Cunha, J.C.F. Telles, and Alvaro L.G.A. Coutinho

Federal University of Rio de Janeiro
Civil Engineering Department / COPPE
P.O. Box 68506 - 21941-972 - Rio de Janeiro - RJ - Brazil
{manoel,telles}@coc.ufrj.br,alvaro@nacad.ufrj.br
http://www.coc.ufrj.br

Abstract. Originally developed by the consortium Sony-Toshiba-IBM for the Playstation 3 game console, the Cell Broadband Engine processor has been increasingly used in a much wider range of applications like HDTV sets and multimedia devices. Conforming the new Cell Broadband Engine Architecture that extends the PowerPC architecture, this processor can deliver high computational power embedding nine cores in a single chip: one general purpose PowerPC core and eight vector cores optimized for compute-intensive tasks. The processor's performance is enhanced by single-instruction-multiple-data (SIMD) instructions that allow to execute up to four floating-point operations in one clock cycle. This multi-level parallel environment is highly suited to applications processing data streams: encryption/decryption, multimedia, image and signal processing, among others. This paper discusses the use of Cell BE to solve engineering problems and the practical aspects of the implementation of numerical method codes in this new architecture. To demonstrate the Cell BE programming techniques and the efficient porting of existing scalar algorithms to run on a multi-level parallel processor, the authors present the techniques applied to a well-known program for the solution of two dimensional elastostatic problems with the Boundary Element Method. The programming guidelines provided here may also be extended to other numerical methods. Numerical experiments show the effectiveness of the proposed approach.

Keywords: Cell Broadband Engine, Boundary Element Method, Boundary Elements, Parallel Programming, Vectorization, SIMD.

1 Introduction

Limitations on power and memory use and processor frequency are leading hardware manufacturers to develop new architectures that are changing the programming paradigms established in the last decades. The performance of a large number of existing serial codes no longer benefits from the rising multi-core

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 490–504, 2008.

technology without a proper porting to these environments. Even parallel applications may need some redesign and rewriting to obtain optimum performance on contemporary microprocessors. One clear example are vectorization techniques much used in the past with vector computers that are now surpassed by the SIMD instructions used in multimedia applications. This particular kind of vectorization differs from old vectorization techniques since it relies on hardware features and extended instruction sets only present on modern processors.

The Cell Broadband Engine is a new architecture that is already playing a significant role in the computing industry in some specific areas [13,14,15] and the knowledge of its strengths and also its current limitations is a decisive factor for engineers and scientists willing to find high-performance solutions to the increasing complexity of their problems and applications. To achieve this goal, this paper introduces the Cell Broadband Engine Architecture and describes in some detail the porting of a well-known serial engineering code to an efficient multi-level parallel implementation.

The implementation of engineering codes, specially using numerical methods to solve elastostatic problems, usually consists in assembling and solving linear equations systems. Even more sophisticated analysis involving elastoplastics or dynamics can be decomposed into a set of such procedures. To generate these systems of equations, numerical methods like finite or boundary elements compute a number of small matrices that are assembled into the equations system accordingly to boundary conditions defined by the problem. In our application, these 2x2 floating-point arrays are specially suited to be computed with SIMD instructions and the paper describes in detail the use of such instructions and how the original algorithm is rewritten to benefit from this vectorization approach.

The text also shows how the proposed algorithm takes advantage of the parallel nature of the Boundary Element Method to efficiently distribute the generation of the equations system among the multiple cores. Since each of the eight computing cores addresses only 256 KB of memory, another challenge to the implementation of engineering codes is the efficient division of the problem - data and code - to fit the memory restraints of these cores. Here, the authors describe the memory transfer mechanisms available on the Cell BE and introduces the use of advanced techniques to hide communication latencies.

The present text is organized as follows: the section 2 presents an outline of the boundary element theory and the following section describes the selected application. Section 4 introduces the Cell Broadband Architecture, its memory transfers mechanisms and the Streaming SIMD Extensions while Section 5 details the multi-level parallel implementation of the code. In section 6 a performance analysis is presented. The paper ends with a summary of the main conclusions.

2 Outline of the Boundary Element Method

The Boundary Element Method (BEM) is a technique for the numerical solution of partial differential equations with initial and boundary conditions [1].

Using a weighted residual formulation, Green's third identity, Betty's reciprocal theorem or some other procedure, an equivalent integral equation can be obtained and converted to a form that involves only surface integrals performed over the boundary. The bounding surface is then divided into elements and the original integrals over the boundary are simply the sum of the integrations over each element, resulting in a reduced dense and non-symmetric system of linear equations.

The discretization process involves selecting nodes on the boundary, where unknown values are considered. Interpolation functions relate such nodes to the approximated displacements and tractions distributions on the respective boundary elements. The simplest case places a node in the center of each element and defines an interpolation function that is constant over the entire element. For linear 2-D elements, nodes are placed at, or near, the end of each element and the interpolation function is a linear combination of the two nodal values. High-order elements, quadratic or cubic, can be used to better represent curved boundaries using three and four nodes, respectively.

Once the boundary solution has been obtained, interior point results can be computed directly from the basic integral equation in a post-processing routine.

2.1 Differential Equation

Elastostatic problems are governed by the well-known Navier equilibrium equation which, using the so-called Cartesian tensor notation, may be written for a domain Ω in the form :

$$G\, u_{j,kk} + \frac{G}{1-2\,\nu}\, u_{k,kj} + b_j = 0 \qquad in \Omega \tag{1}$$

subject to the boundary conditions :

$$\begin{aligned} u &= \bar{u} \qquad on\ \Gamma_1 \qquad and \\ p &= \bar{p} \qquad on\ \Gamma_2 \end{aligned} \tag{2}$$

where u are displacements, p are surface tractions, $\bar{u}$ and $\bar{p}$ are prescribed values and the total boundary of the body is $\Gamma = \Gamma_1 + \Gamma_2$. G is the shear modulus, ν is Poisson's ratio and b_j is the body force component. Notice that the subdivision of Γ into two parts is conceptual, i.e., the same physical point of Γ can have the two types of boundary conditions in different directions.

2.2 Integral Equation

An integral equation, equivalent to Eqs. (1) and (2), can be obtained through a weighted residual formulation or Betty's reciprocal theorem. This equation, also known as Somigliana's identity for displacements, can be written as :

$$u_i(\xi) = \int_\Gamma u^*_{ij}(\xi, x)\, p_j(x)\, d\Gamma(x) - \int_\Gamma p^*_{ij}(\xi, x)\, u_j(x)\, d\Gamma(x) \tag{3}$$

where $b_i = 0$ was assumed for simplicity and the starred tensors, u^*_{ij} and p^*_{ij}, represent the displacement and traction components in the direction j at the field point x due a unit load applied at the source point ξ in i direction.

In order to obtain an integral equation involving only variables on the boundary, one can take the limit of Eq. (3) as the point ξ tends to the boundary Γ. This limit has to be carefully taken since the boundary integrals become singular at ξ. The resulting equation is :

$$c_{ij}(\xi)\ u_j(\xi) + \int_\Gamma p^*_{ij}(\xi, x)\ u_j(x)\ d\Gamma(x) = \int_\Gamma u^*_{ij}(\xi, x)\ p_j(x)\ d\Gamma(x) \qquad (4)$$

where the coefficient c_{ij} is a function of the geometry of Γ at the point ξ and the integral on the left is to be computed in a Cauchy principal value sense.

2.3 Discretization

Assuming that the boundary Γ is discretized into N elements, Eq. (4) can be written in the form :

$$c_{ij}\ u_j + \sum_{k=1}^{N} \int_{\Gamma_k} p^*_{ij}\ u_j\ d\Gamma = \sum_{k=1}^{N} \int_{\Gamma_k} u^*_{ij}\ p_j\ d\Gamma \qquad (5)$$

The substitution of displacements and tractions by element approximated interpolation functions in Eq. (5) leads to :

$$\mathbf{c_i\ u_i} + \sum_{\mathbf{k=1}}^{\mathbf{N}} \mathbf{h\ u} = \sum_{\mathbf{k=1}}^{\mathbf{N}} \mathbf{g\ p} \qquad (6)$$

which can be rearranged in a simpler matrix form :

$$\mathbf{H\ u = G\ p} \qquad (7)$$

By applying the prescribed boundary conditions, the problem unknowns can be grouped on the left-hand side of Eq. (7) to obtain a system of equations ready to be solved by standard methods.

This system of linear equations can be written as :

$$\mathbf{A\ x = f} \qquad (8)$$

where $\mathbf{A}$ is a dense square matrix, vector $\mathbf{x}$ contains the unknown tractions and displacements nodal values and vector $\mathbf{f}$ is formed by the product of the prescribed boundary conditions by the corresponding columns of matrices $\mathbf{H}$ and $\mathbf{G}$. Note that Eq. (8) can be assembled directly from the elements $\mathbf{h}$ and $\mathbf{g}$ without need to generate first Eq. (7).

2.4 Internal Points

Since Somigliana's identity provides a continuous representation of displacements at any point $\xi \in \Omega$, it can also be used to generate the internal stresses. The discretization process, described above, can also be applied now in a post-processing routine.

3 The Application Program

The program reviewed here is a well-known Fortran code presented by Telles [1] for the solution of two dimensional elastostatic problems using linear elements.

The main program defines some general variables and arrays, integer and real, as shown below :

```
program MAIN

  integer :: NN,NE,NP,IPL,INFB,NT,NN2,info
  integer,parameter         :: NMAX=4000
  integer,dimension(NMAX)   :: IDUP
  integer,dimension(NMAX*2) :: IFIP,ipiv
  integer,dimension(NMAX,2) :: INC

  real :: E,PO,C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11
  real,dimension(NMAX)          :: X,Y,C
  real,dimension(NMAX*2)        :: P,XM
  real,dimension(NMAX*2,NMAX*2) :: A

  ! data input
  call INPUT
  ! compute matrix A and independent term XM
  call MATRX
  ! solve system of equations
  call SGESV ( NN2, 1, A, NMAX*2, ipiv, XM, NMAX*2, info )
  if ( info == 0 ) then
    ! output results
    call OUTPT
  else
    write (*,*) 'SGESV : ',info
  endif

end program MAIN
```

The `INPUT` routine reads the program data, the `MATRX` routine computes matrix **A** and the right hand side vector **f**, stored in vector **XM**, while the `OUTPT` routine prints the boundary solution, computes and prints boundary stresses and internal displacements and stresses. The original `SLNPD` subroutine is here replaced by the LAPACK solver `SGESV` [2] which is being ported for the Cell BE processor [8,9].

Subroutine `MATRX` generates the system of equations by assembling directly matrix **A** without creating the global **H** and **G** matrices. This is done by considering the prescribed boundary conditions for the node under consideration before assembling. The leading diagonal submatrices corresponding to **H** are calculated using rigid body translations. Consequently, when the boundary is unbounded a different type of rigid body consideration needs to be applied.

The element influence coefficients are computed calling subroutine `FUNC`. This routine computes all the element integrals required for the system of equations, internal displacements and internal stresses. Numerical integrals are performed over non-singular elements by using Gauss integration. For elements with the singularity at one of its extremities the required integrals are computed analytically to obtain more accurate results.

The boundary stresses are evaluated using subroutine `FENC` that employs the interpolated displacements and tractions to this end. Here, the contribution of

adjacent elements to the common boundary nodes is automatically averaged for non-double nodes. The internal displacements and stresses are obtained by integrating over the boundary elements using subroutine `FUNC`.

The solver is usually the most time consuming routine in BEM programs and various studies have been published on this matter [2]. However, the generation of the equations system as well as the computing of internal points together can take the most part of the processing time [5] and demand special care. While many high-performance parallel solvers are available from standard libraries [2], those two procedures are usually implemented by the researcher and greatly limit the speedup if not optimized. Hence, the Cell BE programming techniques are here applied to the generation of the system of equations and the evaluation of internal point displacements and stresses can also be implemented with the same techniques.

4 The Cell Broadband Engine

The Cell Broadband Engine is a new architecture that succeeds the well-known PowerPC architecture. The Cell BE processor joins in a single chip one PowerPC Processor Element (PPU) and eight Synergistic Processor Elements (SPU). While the PPU runs the operating system and usually the control thread of an application, the SPUs are independent processors optimized to execute data-intensive routines and threads.

At the time of this writing, software for Cell BE is written with C/C++ compilers with vector/SIMD multimedia extensions. However, different SIMD instructions sets for the PPU and SPUs force the programmer to compile separated objects (code modules) in a Cell BE application. Indeed, in a high-level language source code, the SIMD intrinsics for the SPEs are not the same for the PPE which are also different from the PowerPC Altivec instructions, even when executing exactly the same operation.

4.1 The Cell BE Memory Model

The Cell BE Architecture implements a radically new memory organization where PPEs and SPEs access memory in different ways. While the PPE accesses the whole system address space, the SPEs can only address its own private memory. Direct memory access (DMA) commands are used to move data between the main memory and the local memory of each SPEs. With no cache or other hardware mechanisms to automatically load code and data when needed, this memory model leaves to the programmer the task of scheduling DMA transfers between the PPE and the eight SPEs efficiently.

Each SPE private memory includes a 256 KB local storage (LS) to be shared by code and data and 128 registers 128-bits wide. One load and store unit handles data transfers between the local storage and the register file while asynchronous DMA transfers are supported by the Memory Flow Controller (MFC). The MFC supports a maximum transfer size of 16 KB and peak performance is achieved

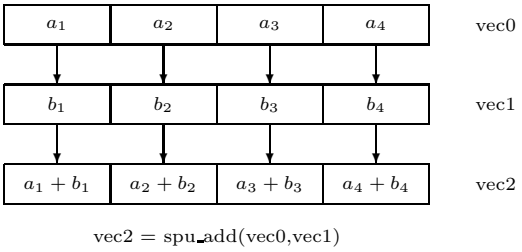


Fig. 1. SIMD Addition

when both the effective (main memory) and LS addresses are 128-bytes aligned and the transfer size is an even multiple of 128.

Besides DMA, Mailboxes is another primary communication mechanism used to exchange queued 32-bits messages. Mailboxes are an useful way to transfer memory addresses and general counters from the PPE to SPEs and can also be used by SPEs to notify the PPE that a memory transfer or computational task has ended.

A third type of communication mechanism, signal notification registers, will not be addressed here. More details on the Cell Broadband Engine Architecture can be found in the literature [10,11,12].

4.2 The Vector/SIMD Multimedia Instructions

Computers were originally classified by Flynn's taxonomy according to instructions and data streams as SISD (single-instruction single-data), SIMD (single-instruction multiple-data), MISD (multiple-instruction single-data) and MIMD (multiple-instruction multiple-data) [6].

As the name suggests, the SIMD model applies to systems where a single instruction processes a vector data set, instead of scalar operands and SIMD instructions perform one operation on two sets of four floating-point single-precision values, simultaneously, as illustrated in Figure 1.

Cell BE provides a large set of SIMD operations. For a full description of all SIMD intrinsic functions the reader is referred to [10]. The implementation of the code here in study with SIMD instructions will be addressed in the next section.

5 The Cell BE Implementation

In the application under study, an equation system is generated in routine `MATRX` with its influence coefficients computed by subroutine `FUNC`. This routine evaluates all the non-singular element integrals using Gauss integration. For elements with the singularity at one of its extremities the required integrals are computed analytically. In the first case, a set of small matrix operations are initially computed, as follows :

$$\begin{bmatrix} UL_{11} & UL_{12} \\ UL_{21} & UL_{22} \end{bmatrix} = -C1 \left[\begin{bmatrix} C2\,logR & 0 \\ 0 & C2\,logR \end{bmatrix} - \begin{bmatrix} DR_{11} & DR_{12} \\ DR_{21} & DR_{22} \end{bmatrix} \right]$$

$$\begin{bmatrix} PL_{11} & PL_{12} \\ PL_{21} & PL_{22} \end{bmatrix} = -C3 \left[\left[\begin{bmatrix} C4 & 0 \\ 0 & C4 \end{bmatrix} + 2 \begin{bmatrix} DR_{11} & DR_{12} \\ DR_{21} & DR_{22} \end{bmatrix} \right] DRDN + C4 \begin{bmatrix} 0 & DRBN_{12} \\ DRBN_{21} & 0 \end{bmatrix} \right] \frac{1}{R}$$

Those 2x2 matrices can be converted into vectors of size 4 and matrix operations can be performed with vector instructions. Thus, a straightforward approach is to use SIMD to evaluate those matrices leaving some intermediate operations to be executed with scalar instructions.

In the original algorithm, those matrices are computed from 2 to 6 times, accordingly to the number of Gauss integration points defined by the chosen integration rule. Alternatively, a fully vector implementation of the matrix computation above can be achieved by using 4 Gauss integration points and evaluating all four values of each coefficient at once, including the intermediate values.

In the application under observation, for each integration point i, the matrix coefficients can be computed as :

$$\begin{aligned}
XMXI_i &= CTE_i * DXY1 + XXS \\
YMYI_i &= CTE_i * DXY2 + YYS \\
R_i &= \sqrt{XMXI_i^2 + YMYI_i^2} \\
DR1_i &= XMXI_i \;/\; R_i \\
DR2_i &= YMYI_i \;/\; R_i \\
UL11_i &= DR1_i^2 - C2 * \log R_i \\
UL22_i &= DR2_i^2 - C2 * \log R_i \\
UL12_i &= DR1_i * DR2_i \\
DRDN_i &= DR1_i * BN1_i + DR2_i * BN2_i \\
PL11_i &= (C4 + 2 * DR1_i^2) * DRDN_i \;/\; R_i \\
PL22_i &= (C4 + 2 * DR2_i^2) * DRDN_i \;/\; R_i \\
PL12_i &= (2 * DR1_i * DR2_i * DRDN_i + C4 * (DR2_i * BN1_i - DR1_i * BN2_i)) \;/\; R_i \\
PL21_i &= (2 * DR1_i * DR2_i * DRDN_i - C4 * (DR2_i * BN1_i - DR1_i * BN2_i)) \;/\; R_i
\end{aligned}$$

Initially using two-dimensional arrays and executed with scalar instructions, the computation presented above - including the intermediate operations - are now performed on vectors and four values are evaluated in each operation. Most of those operations can be performed with basic memory and arithmetic SIMD instructions introduced in the previous section. An SIMD implementation of the vector computation being discussed is presented in Listing 1.

For each integration point i, **UL** and **PL** are used to compute two other matrices, **G** and **H** :

$$\begin{bmatrix} G_{11} & G_{12} & G_{13} & G_{14} \\ G_{21} & G_{22} & G_{23} & G_{24} \end{bmatrix} = \begin{bmatrix} G_{11} & G_{12} & G_{23} & G_{24} \\ G_{21} & G_{22} & G_{23} & G_{24} \end{bmatrix} + \left[\begin{bmatrix} UL_{11}^i & UL_{12}^i \\ UL_{21}^i & UL_{22}^i \end{bmatrix} * B_1^i \begin{bmatrix} UL_{11}^i & UL_{12}^i \\ UL_{21}^i & UL_{22}^i \end{bmatrix} * B_2^i \right] * W^i$$

$$\begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{14} \\ H_{21} & H_{22} & H_{23} & H_{24} \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{23} & H_{24} \\ H_{21} & H_{22} & H_{23} & H_{24} \end{bmatrix} + \left[\begin{bmatrix} PL_{11}^i & PL_{12}^i \\ PL_{21}^i & PL_{22}^i \end{bmatrix} * B_1^i \begin{bmatrix} PL_{11}^i & PL_{12}^i \\ PL_{21}^i & PL_{22}^i \end{bmatrix} * B_2^i \right] * W^i$$

Each one of the 2x4 matrices above can be splitted into two 2x2 matrices, as sampled below :

$$\begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} + \begin{bmatrix} UL_{11}^i & UL_{12}^i \\ UL_{21}^i & UL_{22}^i \end{bmatrix} * B_1^i * W^i$$

Since all values of **UL** are stored in vectors, it is quite simple to perform the multiplications of each value by the respective four values stored in B_1 and W.

Listing 1.

```
DXY1  = spu_splats(DXY[0]);        //            DXY1
DXY2  = spu_splats(DXY[1]);        //            DXY2
tmp0  = spu_splats(xxs);           //                  X[II] - XS
tmp1  = spu_splats(yys);           //                  Y[II] - YS
XMXI  = spu_madd(CTE,DXY1,tmp0);   // .5 (XI + 1) DXY1 + X[II] - XS
YMYI  = spu_madd(CTE,DXY2,tmp1);   // .5 (XI + 1) DXY2 + Y[II] - YS
tmp2  = spu_mul(YMYI,YMYI);        //               YMYI^2
tmp3  = spu_madd(XMXI,XMXI,tmp2); //      XMXI^2 + YMYI^2
INVR  = rsqrtf4(tmp3);             // sqrt(XMXI^2 + YMYI^2)
DR1   = spu_mul(XMXI,INVR);        // XMXI / R
DR2   = spu_mul(YMYI,INVR);        // YMYI / R
LOGR  = logf4(INVR);               //                 log R
BN2   = spu_splats(BN[1]);         //                  BN2
UL12  = spu_mul(DR1,DR2);          //    DR1 DR2
DR11  = spu_mul(DR1,DR1);          //    DR1^2
DR22  = spu_mul(DR2,DR2);          //    DR2^2
tmp4  = spu_mul(DR2,BN2);          //           DR2 BN2
tmp5  = spu_mul(DR1,BN2);          //                                DR1 BN2
BN1   = spu_splats(BN[0]);         //       BN1
UL11  = spu_madd(C2v,LOGR,DR11);   // DR1^2 + C2 log R
UL22  = spu_madd(C2v,LOGR,DR22);   // DR2^2 + C2 log R
tmp6  = spu_madd(DR11,TWO,C4v);    //  2 DR1^2 + C4
tmp7  = spu_madd(DR22,TWO,C4v);    //  2 DR2^2 + C4
tmp8  = spu_add(UL12,UL12);        //  2 DR1 DR2
tmp9  = spu_msub(DR2,BN1,tmp5);    //                     DR2 BN1 - DR1 BN2
DRDN  = spu_madd(DR1,BN1,tmp4);    // DR1 BN1 + DR2 BN2
tmp10 = spu_mul(tmp6,DRDN);        // (2 DR1^2 + C4) DRDN
tmp11 = spu_mul(tmp7,DRDN);        // (2 DR2^2 + C4) DRDN
tmp12 = spu_mul(tmp9,C4v);         //                  C4 (DR2 BN1 - DR1 BN2)
PL11  = spu_mul(tmp10,INVR);       // (2 DR1^2 + C4) DRDN / R
PL22  = spu_mul(tmp11,INVR);       // (2 DR2^2 + C4) DRDN / R
tmp13 = spu_msub(tmp8,DRDN,tmp12); //  2 DR1 DR2 DRDN - C4 (DR2 BN1 - DR1 BN2)
tmp14 = spu_madd(tmp8,DRDN,tmp12); //  2 DR1 DR2 DRDN + C4 (DR1 BN2 - DR2 BN1)
PL21 = spu_mul(tmp13,INVR);        // (2 DR1 DR2 DRDN - C4 (DR1 BN2 - DR2 BN1)) / R
PL12 = spu_mul(tmp14,INVR);        // (2 DR1 DR2 DRDN + C4 (DR1 BN2 - DR2 BN1)) / R
```

However, there is no SIMD instruction to perform the sum of the elements of a vector needed in the computation of **G**. Using the SIMD shuffle instructions, the values stored on four vectors can be reordered to obtain the same effect of a matrix transposition, although here the operations are performed on vectors. A possible SIMD implementation of the computations just presented is presented in Listing 2.

Well-known optimization techniques usually applied to scalar codes can also be used in the implementation of vector algorithms in order to replace long latency instructions and to reduce data dependence. Data dependence is the major obstacle to the vectorization of any algorithm. Even well-written programs enclose data dependencies due to the nature of the applications. High performance techniques are presented by the authors in previous studies [3,4] and will not be addressed here.

The boundary element method has a parallel nature since each boundary node generates two rows on the equations systems. The computing of each pair of rows is totally independent and can safely be performed concurrently. Hence, a straightforward approach is to distribute the boundary elements equally between the eight SPEs. The same procedure can also be used to the computing of internal

points. With no communications needed between the SPEs and taking in account that each boundary node must be integrated with all elements on the boundary, an efficient approach is to leave to the SPEs the task of moving and processing data while the PPE only starts the same thread on each SPE, transferring the global addresses of input data vectors and arrays, as demonstrated in Listing 3.

Listing 2.

```
vector unsigned char permvec1 =
{0,1,2,3,16,17,18,19,4,5,6,7,20,21,22,23}; vector unsigned char
permvec2 = {8,9,10,11,24,25,26,27,12,13,14,15,28,29,30,31}; vector
unsigned char permvec3 =
{0,1,2,3,4,5,6,7,16,17,18,19,20,21,22,23}; vector unsigned char
permvec4 = {8,9,10,11,12,13,14,15,24,25,26,27,28,29,30,31}; ...
tmp0  = spu_mul(UL11,B1W); tmp1  = spu_mul(UL12,B1W); tmp2  =
spu_mul(UL22,B1W); tmp3  = spu_shuffle(tmp0,tmp1,permvec1); tmp4
= spu_shuffle(tmp0,tmp1,permvec2); tmp5  =
spu_shuffle(tmp1,tmp2,permvec1); tmp6  =
spu_shuffle(tmp1,tmp2,permvec2); tmp7  =
spu_shuffle(tmp3,tmp5,permvec3); tmp8  =
spu_shuffle(tmp3,tmp5,permvec4); tmp9  =
spu_shuffle(tmp4,tmp6,permvec3); tmp10 =
spu_shuffle(tmp4,tmp6,permvec4); tmp11 = spu_add(tmp7,tmp8); tmp12
= spu_add(tmp9,tmp10); tmp13 = spu_add(tmp11,tmp12); Cv    =
spu_splats(C); CC1   = spu_mul(Cv,C1v); G1    =
spu_mul(tmp13,CC1); ...
```

In each SPU, the SPU number (spu_id) is read from the PPU using mailbox and the input data is transfered from main memory to local arrays using DMA. In this implementation, the boundary nodes are evenly distributed among the SPUs and only a pair of rows of the equations system corresponding to each node is computed and transfered from the local storage to the main memory in each iteration, as shown in Listing 4.

The concurrent computing of a given pair of rows with the asynchronous DMA transfer of the previous pairs of rows is a technique used to hide memory transfer latencies, known as double-buffering.

Due to 256 KB local storage size, the initial approach of loading all the input data in SPU's local arrays limits the number of nodes to approximately 4000. In an alternative implementation, only parts of each input array can be transfered from the PPU to the SPU. Although increasing the number of DMA transfers, this technique reduces the memory size demand and increases the maximum number of the nodes to be processed. Using 1 GB main memory (QS20), the total number of nodes is limited to 6000 while in a 2 GB system (QS21) it is limited to 10000, approximately.

6 Results

The parallel implementation presented here run on a Cell Blade QS21 server with two processors and 2 GB main memory shared between the processors.

Listing 3.

```
typedef struct {
  float *X;
  ...
} BESTRUCT;
float X[NMAX] __attribute__((aligned(128)));
...
BESTRUCT bestruct __attribute__((aligned(128)));
bestruct.X = X;
...
extern spe_program_handle_t bizep_spu;
int main(void) {
  ppu_pthread_data_t ppdata[8];
  for (i=0;i<NSPU;i++) {
    ppdata[i].context = spe_context_create(0,NULL);
    spe_program_load(ppdata[i].context,&bizep_spu);
    ppdata[i].entry = SPE_DEFAULT_ENTRY;
    ppdata[i].argp = (void *) &bestruct;
    ppdata[i].envp = (void *) 128;
    pthread_create(&ppdata[i].pthread,NULL,&ppu_pthread_function,&ppdata[i]);
    spe_in_mbox_write(ppdata[i].context,&i,1,SPE_MBOX_ANY_NONBLOCKING);
  }
  for (i=0;i<NSPU;i++) {
    pthread_join(ppdata[i].pthread,NULL);
    spe_context_destroy(ppdata[i].context);
  }
  printf ("End of PPU thread\n");
  return 0;
}
```

Listing 4.

```
// the MATRX routine is now the main function running on the SPE
int main(unsigned long long speid,unsigned long long argp,unsigned long long envp) {
  BESTRUCT bestruct __attribute__((aligned(128)));
  // read the SPU id using mailbox
  unsigned int spu_id = spu_read_in_mbox();
  // transfer the structure data from PPU to SPU using DMA
  int tag = 1, tag_mask = 1<<tag;
  mfc_get(&bestruct,(unsigned int) argp,envp,tag,0,0);
  mfc_write_tag_mask(tag_mask);
  mfc_read_tag_status_all();
  // transfer vectors and arrays using DMA
  mfc_get((char *) X,(unsigned long int) bestruct.X,16384,tag,0,0);
  mfc_read_tag_status_all();
  ...
  for (i=first_node;i<=last_node;i++) { // loop over boundary nodes
    for (j=1;j<=bestruct.NE;j++) { // loop over boundary elements
      FUNC(ICOD,C,II,IF,XS,YS,G,H); // SIMD routine
      ...
    }
    // transfer local array A using DMA
    ppu_address = (unsigned long int) (i-1) * sizeof(A);
    for (j=0;j<4;j++)
      mfc_put((char *) A+j*16384,(unsigned long int) ppu_address+j*16384,16384,tag,0,0);
    mfc_read_tag_status_all();
  }
  return 0;
}
```

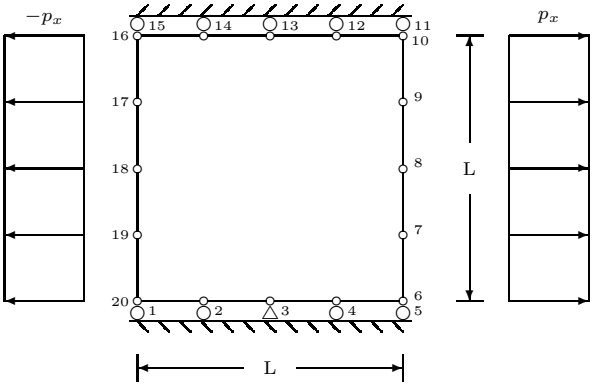


Fig. 2. A square plate under biaxial load

Table 1. QS21 Results - 4000 nodes

SPUs	1	2	4	8
real	7.617s	3.822s	1.926s	0.978s
user	0.002s	0.002s	0.002s	0.002s
sys	0.148s	0.151s	0.159s	0.171s
Speedup	-	1.993	3.955	7.788

Each 3.2 GHz processor has a 64-bits PowerPC with two 32 KB L1 caches and a 512 KB L2 cache and eight SPUs with 256 KB memory each. The operating system is Linux Fedora 7 with Cell BE SDK 3.0.

The study case to be presented here corresponds to a square plate under biaxial load, as found in [1]. The schematic description of the problem is depicted in Figure 2.

The results shown on Table 1 refer to the generation of a 8000x8000 equations system while Table 2 refers to a 20000x20000 equations system of single-precision floating-point elements . The use of a smaller number of SPUs is presented here only for sake of reference, since in practice there is no sense to leave a vector core idle. Also for sake of reference, the results of the SIMD implementation [6,7] of the same code on another architecture, a quadcore Intel Xeon 2.66 GHz (X5355) processor with 8 GB memory, is presented on Tables 3 and 4.

Table 2. QS21 Results - 10000 nodes

SPUs	1	2	4	8
real	3m4.297s	1m32.350s	46.252s	23.245s
user	0.002s	0.002s	0.002s	0.003s
sys	0.887s	0.901s	0.920s	1.098s
Speedup	-	1.996	3.985	7.928

Table 3. Intel Xeon Results - 4000 nodes

time	Original	Autovectorization	SSE Intrinsics
real	5.132	3.604	1.992
user	0.212	0.152	0.196
sys	0:5.35s	0:3.75	0:2.18

Table 4. Intel Xeon Results - 10000 nodes

time	Original	Autovectorization	SSE Intrinsics
real	45.822	32.918	12.880
user	1.184	0.956	1.160
sys	0:47.02	0:33.88	0:14.06

The almost linear speedups shown in Tables 1 and 2 show the effectiveness of the algorithm used here and emphasize the parallel nature of the Boundary Element Method. The technique of distributing the boundary nodes between the SPUs can also be used to distribute workload between the cores of a blade and among multiple blades. The same approach is used in the shared and distributed memory implementations of this and other BEM codes [4,5] and will not be discussed here.

It must be noticed that the results shown in Table 1 refers to an implementation where all the input data are loaded to the SPE local store while Table 2 refers to an implementation where only parts of input data are transfered during the runtime, as explained in the previous section. In the first case, most of DMA transfers (99%) are used to write the equations system into main memory. In the second case, to bypass the SPE local store size limitation, most of DMA transfers (92%) are performed to load the input data into SPE's local store. A radical change in the input data layout could reduce DMA reads and will be implemented in a subsequent work.

7 Conclusions

The Cell Broadband Engine processor is a new architecture developed originally to be used in game consoles and multimedia devices. To face the current limitations on power and memory use and processor frequency, the Cell Broadband Engine introduces a multi-core processor with a highly innovative memory model. As one of the many options of a changing industry, this paper addresses the viability of this environment to run engineering codes, specially numerical methods applications.

Here, the basic aspects of Cell BE architecture and its programming techniques are presented with the porting of a well-known boundary element code to solve two-dimensional elastostatic problems. As shown, existing codes can be rewritten to run on Cell BE after a careful change of the serial algorithm in order to benefit from the multiple vector cores. The results presented here show

the effectiveness of the proposed algorithm and emphasize the parallel nature of Boundary Elements. The same parallelization technique can be used to distribute the workload between the SPUs, the cores of a Cell BE blade or among multiple blades. At the time of this writing, these results clearly show the Cell BE well suited to run the kind of engineering application presented here.

However, some current limitations of Cell BE must be taken in account. The first implementation of this family of processors is designed to handle efficiently single-precision floating-point operations while double-precision are usually ten times slower. With no cache and other hardware mechanism developed to handle the processor-memory performance gap, the Cell BE leaves to the programmer the task of scheduling data transfers between main memory and local storages efficiently. This radical design leads to greater learning and programming efforts. A very limited amount of memory in each vector core also implies in significant changes on existing algorithms resulting in increasing development costs and loss of portability.

With the implementation of efficient double-precision floating-point operations, larger memory, a greater number of vector cores and a set of development tools, the next generations of Cell Broadband Engine will play a major role in the computer industry in the near future and become one of the main options for engineering and scientific applications.

Acknowledgments. The authors are in debt to Dr. Michael Perrone, Dr. Ulisses Mello and IBM T.J. Watson Research Center for the support in this work. Hardware resources were provided by IBM Innovation Center, Dallas. M.T.F. Cunha is supported by a CAPES grant from the Ministry of Education, Brazil.

References

1. Brebbia, C.A., Telles, J.C.F., Wrobel, L.C.: Boundary Elements Techniques: Theory and Applications in Engineering. Springer, Berlin (1984)
2. Dongarra, J., et al.: LAPACK User's Guide, 3rd edn. SIAM, Philadelphia (1999)
3. Cunha, M.T.F., Telles, J.C.F., Coutinho, A.L.G.A.: On the Parallelization of Boundary Element Codes Using Standard and Portable Libraries. Engineering Analysis with Boundary Elements 28/7, 893–902 (2004)
4. Cunha, M.T.F., Telles, J.C.F., Coutinho, A.L.G.A.: A Portable Implementation of a Boundary Element Elastostatic Code for Shared and Distributed Memory Systems. Advances in Engineering Software 37/7, 893–902 (2004)
5. Cunha, M.T.F., Telles, J.C.F., Coutinho, A.L.G.A.: Parallel Boundary Elements: A Portable 3-D Elastostatic Implementation for Shared Memory Systems. In: Daydé, M., Dongarra, J., Hernández, V., Palma, J.M.L.M. (eds.) VECPAR 2004. LNCS, vol. 3402, pp. 514–526. Springer, Heidelberg (2005)
6. Cunha, M.T.F., Telles, J.C.F., Ribeiro, F.L.B.: Streaming SIMD Extensions Applied to Boundary Element Codes. Advances in Engineering Software (2008), doi: 10.1016/j.advengsoft.2008.01.003

7. Cunha, M.T.F., Telles, J.C.F.: On The Vectorization of Engineering Codes Using Multimedia Instructions. Engineering Analysis with Boundary Elements (2008) (under revision)
8. Kurzak, J., Buttari, A., Dongarra, J.: Solving Systems of Linear Equations on the Cell Processor Using Cholesky Factorization. LAPACK Working Note 184, CS-UTK Tech. Report 07-596
9. Kurzak, J., Dongarra, J.: Implementation of the Mixed Precision in Solving Systems of Linear Equations on the Cell Processor. LAPACK Working Note 177, CS-UTK Tech. Report 06-580. Concurrency: Practice and Experience 19/10, 1371–1385 (2007)
10. Cell Broadband Engine Programming Tutorial. IBM (2007)
11. Cell Broadband Engine Programming Handbook. IBM (2007)
12. C/C++ Language Extensions for Cell Broadband Engine Architecture. IBM (2007)
13. Liu, Y., Jones, H., Vaidya, S., et al.: Speech Recognition Systems on the Cell Broadband Engine Processor. IBM Journal of Research and Development 51/5, 583–591 (2007)
14. Kachelrieb, M., Knaup, M., Bockenbach, O.: Hyperfast Parallel-beam and Cone-beam Backprojection Using the Cell General Purpose Hardware. Medical Physics 34/4, 1474–1486 (2007)
15. Bockenbach, O., Mangin, M., Schuberth, S.: Real Time Adaptive Filtering for Digital X-ray Applications. In: Larsen, R., Nielsen, M., Sporring, J. (eds.) MICCAI 2006. LNCS, vol. 4190, pp. 578–587. Springer, Heidelberg (2006)

Grid Data Management: Open Problems and New Issues from the P2P Perspective

Esther Pacitti

INRIA and LINA, University of Nantes
Esther.Pacitti@univ-nantes.fr

Abstract. Initially developed for the scientific community, Grid computing is now gaining much interest in important areas such as enterprise information systems. This makes data management critical since the techniques must scale up while addressing the autonomy, dynamicity and heterogeneity of the data sources. In this paper, we discuss the main open problems and new issues related to Grid data management. We first recall the main principles behind data management in distributed systems and the basic techniques. Next we make precise the requirements for Grid data management and introduce the main techniques needed to address these requirements mainly by using P2P techniques. To illustrate the use of these techniques that may be implemented in Grid infrastructures, we present a peer to peer multi-master replication approach for collaborative applications, and continuous stream processing algorithms built over distributed hash tables using gossiping.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, p. 505, 2008.

Data Management Concerns in a Pervasive Grid

Jean-Marc Pierson

IRIT, UMR CNRS
pierson@irit.fr

Abstract. In this article, we give our vision of a Pervasive Grid: A grid dealing with light, mobile and uncertain devices, using context-awareness for delivering the right information using the right infrastructure to final users. We focus on the data side of the problem, since it encompasses a number of problems related to such an environment, from data access, query optimization, data placement, to security, adaptation,...

Keywords: Grid computing, pervasive systems, data management.

1 Introduction

The last decade have seen the wide development and adoption of Grid Computing [16],[18]. Its main purpose is to help for coordinating large-scale heterogeneous resources sharing (hardware, software) and problem solving in dynamic, multi-institutional virtual organizations. From its original purpose of distributed supercomputing and massive data processing, Grid computing moves more and more to become a common platform and way for utility and service computing. It allows the sharing and the coordination in heterogeneous environments (from clusters, main frames to personal computers) through relatively well developed middlewares architectures like OGSA [17] and one major implementation Globus [1].

On the other hand, mobile, small, wearable personal devices are entering our everyday life. People expect more and more information and services to be accessible or readable on their hand-held digital assistant, from cell phones to personal digital assistants. Pervasive Information Systems aims at offering the right information, to the right people, at the right time, on the right format. People are not aware of the transparent infrastructure that offers them the information or the service. Non intrusiveness, mobility, context-awareness and unpredictability characterize the key constraints of these environments.

The current trend is to interconnect the two domains, offering the enormous potential for resource sharing to hand-held devices. Grid services would then be accessible, controllable from pervasive devices, and results would be sent to these devices for on-site or remote analysis.

Our purpose is a little bit different, and go a step further : We envision a Pervasive Grid [28]. In this Pervasive Grid, resource sharing is still the key concept, and we include the full heterogeneity of devices (even hand-helds) in this Grid. While I. Foster and al. denote in [16] "A computational grid is a hardware and

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 506–520, 2008.

software infrastructure that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities", the pervasive term has been foreseen here as a way to offer a transparent access for high computational resources for end-users. To paraphrase I. Foster, a Pervasive Grid is a hardware and software infrastructure that provides dependable, consistent and inexpensive access to pervasive resource sharing capabilities. Users are not aware of the set of available resources, they just use services without notice. These resources may be either computational clusters, mobile phones, or whatever can share something with the outside world. Conversely, in a proactive way, the pervasive environment may provide users with information according to their preferences.

In the following, P-Grid will refer to Pervasive Grid.

Applications taking advantage of a P-Grid can be categorized in three groups:

- Opportunistic applications: These applications will discover and use pervasive information and resources to potentially adapt, optimize, improve QoS. As an example, vehicle safety systems may use information from other vehicles to warn about hazards.
- Cooperative applications: These use multiple application entities (providing subset of service or information) allowing for making decision in an autonomous manner. For instance group of cars can share information to estimate global traffic
- Control Applications: These provide autonomic control capabilities using actuators to impact their environment, e.g. a car may anticipate road conditions to use the brakes.

P-Grids environments are inherently large, heterogeneous, dynamic (mobility, context), aggregating a large number of potentially unknown and constrained entities (energy, CPU power, HD space, security) with a potential for high computations and/or tremendous amount of data.

Key aspect is uncertainty. Uncertainty is found as several levels:

- Systems: in their structure (flat, P2P,), dynamism (entities may enter or leave at any time), components (in terms of reliability, capabilities, costs, energy), constituents (number, protocols, locations,)
- Applications: dynamism in their compositions, couplings and interactions (opportunistically connecting)
- Information: quality, availability, placement, trust, compliance with common understanding and semantics

The need to adapt to changes in the environment or context is mandatory. Applications should be constructed from discrete, self managing components with separate functional, non-functional (QoS, reliability,costs), interaction-coordination behaviors. Semantic knowledge and autonomous mechanisms must be enforced at the middleware and application level. As a general rule, systems and middleware should support context-/content-/location-awareness, with dynamic data-/knowledge and time constraints for the executions, the adaptations and the compositions of services and application while guaranteeing reliable and resilient execution and predictable performance.

As one can imagine, inventing such a Pervasive Grid is a large task, that can not be described in one chapter. We will focus our purpose in this article on the data side of the problem : data management in a Pervasive Grid. The first reason for this is that data is the core of the system: Most of the applications use data as input or produce data as output. The second reason is because the data management has long been forgotten in Grid Computing, and efforts has been made late in order to handle the available data. And finally, the data side exhibits a number of problems symptomatic of the Pervasive Grid environment.

The rest of the chapter is organized as follows : After motivating the lacks in existing systems (either Pervasive or Grid Computing) to achieve our vision in section 2, we detail in section 3 who are the actors of a Pervasive Grid in a global architecture view, and we identify a number of standalone services for data management one would like to integrate. We present in section 4 mutual links, interactions, and today advances in our group with the most promising steps in the direction of a Pervasive Grid. Before concluding, we discuss some implementation issues in section 5.

2 Related Work

The two domains related to a Pervasive Grid are the Grid and the Pervasive systems, which are both very young research fields. Nevertheless, due to both very active communities, some works have already started in the direction of a P-Grid.

Davies, Storz and Friday [15][39] were the first to introduce the concept of "Ubiquitous Grid", that is close to our P-Grid. The purpose of their article is to compare the notion of Grid Computing (definition of I. Foster [17]) and the notion of Pervasive Systems (definition of M. Weiser [40]). They identify similar interests : heterogeneity, interoperability, scalability, adaptability and fault tolerance, resources management, services composition, discovery, security, communication, audit, payment. They present briefly an use case developed with Globus Toolkit 3 (GT3), for an ubiquitous grid and their solution. Lack of details makes it difficult to evaluate exactly what has been done to make GT3 behave like an ubiquitous grid, and what parts of ubiquity has been addressed. Moreover, the authors do not deal with data management.

Hingne et al. [22] propose a multi-agent approach to realize a P-Grid. They are interested in communication, heterogeneity, discovery and services composition, and scheduling of tasks between the different devices constituting the P-Grid. Almost nothing is said concerning the data management.

McKnight et al. [27] introduce the concept of Wireless Grid. Their interest is on the mobile and nomadic issues that they compare with traditional computing grids, P2P networks and web services. An interesting point of this article is to put in light the relationships between these actors. The authors focus their interest on services they identify as the most important: resources description and discovery, coordination, trust management and access control. These

latter points are related to data management, together with the resources (data) description and discovery.

In [37], S.H. Srinivasan details a Wireless Pervasive Grid architecture. The author separates its grid in two parts: One is the "backbone grid", physically linked like the network backbones; the other one is wireless, the "access grid". Agents realize the proxy between the two grids, and behave on behalf of the mobile devices of the "access grid" on the "backbone grid". Interesting points in this proposal concern the pro-activity and the context-awareness of the presentation to the end-users.

Coulson et al. [14] present a middleware structured using a lightweight runtime component model (OpenCom) that enables appropriate profiles to be configured on a wide rage of device types, and facilitates runtime reconfiguration (as required to adapt to dynamic environments).

More recently, Coronato and De Pietro [13] describe MiPEG, a middleware consisting of a set of services (compliant to grid standard OGSA) enhancing classic Grid environments (namely the Globus Toolkit) with mechanisms for handling mobility, context-awareness, users' session and distribution of tasks on the users' computing facilities.

Complementary to these, some other works exist to tackle some aspects of pervasive systems in the computing grids, mainly mobility and adaptation to light devices. Some works [3,21,19] have focused on the use of light devices to interact with computing grids, submitting jobs and visualizing results. Integrating more the mobile devices in the grids are [30,29] that propose proxy services to distribute and organize the jobs among a pool of light devices. [25] solicits surrounding devices to participate in a problem solving environment. Others [2,24] are interested in the advantages of mobility features of IPv6 in the notification and adaptation of grids. Mobile agents are used in [8,5] to migrate objects between sites. Some researchers [11,12] have investigated how a grid middleware (Legion, OGSI.NET) can be adapted to tackle mobility issues. Context-awareness are basis for the works of [42], while the authors in [41] include mobility and context-awareness in their approach. Context-awareness is the primary focus of the work presented in [23]. The authors present an extension of virtual organization to context, providing personalization of the services.

In the context of data management, [10] was one of the first to examine the problems inherent to pervasive computing related with data management. In the Grid computing field, Stockinger et al. [38] have exhibited in the framework of the european DataGrid project the major issues a data management toolkit should address. These papers served as basis for our work.

Unfortunately, almost none of these works handle the complexity of a P-Grid as a whole, dealing with mobility, context-awareness and uncertainty, at the same time. And none are handling the data management itself. We thus propose in the following to have an integrated view of services related to data management in a Pervasive Grid.

3 Services for Data Management in a P-Grid

3.1 Data

The first thing is to identify which data a P-Grid can deal with. We have identified 3 different kinds of data:

- User data: These include personal data (for instance stored on personal handheld) or business data (stored on high-end storage)
- Application data (or service data): These data are produced by an application, and represent its output.
- Acquired data: These are coming from sensors and help either to construct the context of the data usage or impact the functions of the middleware system.

These data differs in terms of size, confidentiality, quality, frequency, operations. These different characteristics lead to different need for data management in terms of storage, reliability, availability, security, sensitiveness and quality. Hence the data quality and information uncertainty management must be tackle appropriately. We foresee thus the need to synthesize information with dynamic properties and properties from data streams, to detect events, to assess data quality, to cluster data, to adapt the frequency and the level of information gathering, to support online, in-network and immediate use and processing

3.2 Services

Data management in a P-Grid shares some characteristics of Grid computing and Pervasive computing. It should share some trends from autonomous computing and semantic web/grid and some ideas coming from P2P computing, Mobile computing, ad-hoc and sensors network, embedded systems.

In the following, we follow a service oriented architecture for data management in a P-Grid, keeping in mind the needs sketched in the previous section as well as the constraints of the environment and the data.

We have identified a minimal set of services related to the management of data, that must be present in a P-Grid. We expose in this section these services, while we will figure out the location of these services in the global architecture, as well as their mutual links in the section 4.

- storage: This service is responsible for the physical storage of the data on the P-Grid, regardless the reliability of the storage site. This service allows one user (or application) to store one piece of data on the system: The underlying storage solution may be a file system, a database or whatever can store data. The user is the source of authority (SOA) of this data, and controls the authorizations of access, replication, ... One can notice that all data may not be in the P-Grid, for performance or security reasons. For instance, it may become impossible to store all data issued by sensors since the throughput could be too large. Some data, like medical data, may never be stored "as is" outside an hospital, for obvious confidentiality reasons and should be anonymized or encrypted beforehand.

- replication: Two main reasons push in favor of the replication service: performance and reliability. Adjusting the number of copies and their location to suit their usages (i.e. moving the replicas around) helps to increase the performance of the system and to decrease the user queries process time. The second motivation comes from the instability of the P-Grid storage sites. Increasing the number of replicas of data allows to increase their availability, but decreases the control over their proliferation and their consistency. This is particularly important when dealing with volatile and mobile devices. Nevertheless, all data can not be replicated, and a judicious choice must be done according to the usage of the data, which must be monitored.
- cache: A cache service allows to keep in temporary storage space some accessed data or results of user queries. This reduces the cost of accessing data (it may be a network cost as well as a financial cost). The main difference with the previous replication service is the hidden property of a cache service, that acts transparently with respect to the users requests. It is a proxy between these and the data themselves. Collaborative caching must be deployed, so that several distributed cache services cooperate, using the semantics of the cached data (notice here that a global agreement of data semantic representations has to be accepted -ontologies can help in this matter). This collaboration increases the total space dedicated to the cache as well as the global hit-rate. The collaboration of the caches can handle the split of files in different caches but then appears the problem of volatility of the devices, making some parts of the data temporary unavailable (thus a replication service have to be coupled to the cache service).
- access: This service has two tasks. First, it accesses the data previously stored by the storage service. Second, it realizes the access to data stored outside the P-Grid, in external databases, flat files... Doing this, it must implement an integration mechanism between the different available data sources.
- indexing: The indexation service is responsible for maintaining a global link between a physical data and a logical representation of this data. This representation may be a logical name, or a more detailed semantic description contained in metadata. This latter, that is a semantic indexing of the data, allows to have a semantic view of the available data. This issue can help to group geographically data (or their replicas) which are dealing with the same themes. A second benefit of a semantic indexing is to optimize the replacement policies of the collaborative caches, keeping in those the hottest topics (the data related to the more accessed themes). Finally, it allows to group efficiently data to apply an adapted access control policy to a set of data.
- search: This service relies on the previous one to find data. But it must also allow for hybrid queries, that act on stored data as well as on computable data (i.e. data that can be extracted or generated from stored data, after processing).
- transport:Data transport service is managing the communication of the data in the P-Grid. It relies on the the local available infrastructure to deliver the data: It can be either FTP-like for classic data, streaming for multimedia

data, it can use UDP or TCP over a UMTS, WiFi, Bluetooth underlying protocol. As multiple transport solutions may coexist on one particular device, this service must choose which combination is the best choice for one communication. Moreover, the data transport capacity might evolve over time as the user is mobile and network settings and quality change continuously.

- security: This service can be decomposed in several parts. Data security is fundamental in a P-Grid, given its high degree of dynamic and unreliability. Allowing one to access a piece of data is first to allow one to enter the device storing the data. This must be controlled via identification, then authentication. This is far from trivial when users can not be known everywhere, and when centralized solutions (like CAs) must not become single points of failure. A high level of decentralization must be achieved. Typically, a Single Sign On must be available in the infrastructure, and trust management procedures have to be enforced. When one is authenticated, then comes the problem of authorization. Data access must be controlled at a fine grain, at the closest to the data itself, to allow the SOA of the data to give minimal sufficient rights, and the sites holding the data must control the given permission on the fly. Splitting the data from the permissions to access the data is a solution to limit the scope of corrupted sites (increasing fault tolerance). Finally, an encryption mechanism must be provided in order to increase the confidentiality and non disclosure of the data on the devices. The communication encryption is the problem of the transport service. *Related to security is also the history service, that will be detailed later, to allow for traceability in the users' usage of the data.*
- adaptation: The adaptation service is a key service for the data in a P-Grid. Before delivering a piece of data to the final user, it must be adapted to the device of this user: First, it may not correspond to the physical characteristics of the device (for instance, the result is an audio file, but no audio output is available on the device; the result should be transformed to plain text). Second, the profile of the user must be taken in consideration (for example, the language of the result should be changed). Finally, the available transport service might not handle the result (if the result is a video, and Bluetooth is the only communication possibility, the bit-rate should be adapted accordingly). Adaptation services must be available on the P-Grid, or external and accessible through the P-Grid. Combining and chaining these services, using their operational semantic is the responsibility of the adaptation service.
- processing: The above adaptation service is a particular processing service. The user may initiate any available service on data, from visualization to anonymisation, for instance. This aspect is not directly related to data and will not be discussed further.
- computer-human interaction: The wills of the user, as well as his feedback regarding the usage of the system should be taken into account for presenting data and computation results. Non intrusiveness and intuition should

drive the research in this field and interaction systems using all modalities of interaction (from visual to sound to tactile) be developed.

These different services must be developed as independent modules, if possible, and must not rely on a specific middleware : Indeed, a hand-held device will not have the same capabilities to embed a heavy middleware compared to an office computer. Only their presentation (API, exchange protocols and messages format) must be inter-operable. We will discuss in section 5 which technologies can be used to achieve these constraints.

These modules must also be developed as distributed and decentralized modules when possible. Scalability issues lead to avoid single point of decision, that could become single point of failure. Providing distributed modules may increase the reliability of the system (if it exists a part of redundancy) and also increase the global performance of the system, balancing the load between several instances of one service. Autonomy and self-organization of the modules must be enforced in their development and management.

To the minimal set of services dedicated to data management, expressed above, we express the need of underlying services, not specific to data management but useful for it.

- execution service: This service is responsible for the execution of services on the P-Grid. It realizes the interface with the scheduler, organizes and verifies the correct execution of the tasks.
- monitoring service: The monitoring service must be aware of the current state of the resources, from services to network to hosts. This service allows two benefits: to have a global view of the P-Grid, in terms of available services (and their state : working, idle, stopped, ...), hosts, network resources, data location, ... and to personalize this view according to the dynamic permissions and needs of the users.
- discovery service: Discovery is important for mobile or transient devices. It permits these devices to know which are the available services in their neighborhood, so that interactions can be initiated.
- history service: This service is a memory of what occurred on the P-Grid. It has two main roles : First, it keeps a trace of the access to the services and the data, for traceability or financial reasons. Second, it allows to reconstruct a composition of services for one user (or a group of users), in a given context, saving some time discovering the services or the data available. Techniques such a data mining in this history can be used for this issue.

The presented architecture is quite different from a Computing Grid, taking into account at each stage of the services development the uncertainty, the instability, the mobility and the context-awareness of some components of the P-Grid. Conversely, it differs from a pure pervasive systems since it can use some stable infrastructure to store data or execute some compute-intensive services. It is not either a Peer2Peer architecture, since each node does not have the same role, and roles may be distributed dynamically.

4 Links, Interactions, and Today First Steps

In a P-Grid, several infrastructures may coexist, some being stable, some being unstable. It is thus necessary to identify which services must be deployed on the stable part, and which services may be deployed on the unstable part. Moreover, all services do not need to be embedded everywhere: Indeed, some devices can not afford to have too much services on them. The placement of the services in the architecture depends on a number of factors: its importance in the global architecture, the importance of its responsiveness, of the quality of the response... Moreover, the properties of these factors evolve with time and the available infrastructure. As an example, if a device is connected to the Internet, it can use distant web services while he can only use its surroundings in the other case.

Finally, to construct a P-Grid and thus the data management in a P-Grid, two approaches are complementary. The first one consists in modifying a normal grid middleware (like Globus for instance in [13]) and to inject the mobility of the actors, the context of the actors, the unpredictability and more importantly uncertainty of the data and systems.

The second one consists in constructing a new adapted modular middleware with some key features that can be summarized here:

- Decentralization: no centralized control, a complete autonomy with locality of processing and data
- Fault tolerance and security.
- Interoperable with existing services.
- Dynamic: Decisions have to adapt to context. Also the need for both reactivity and pro-activity.
- Semantics: protocols, algorithms, taking into account the semantics of the data, the accesses and the services
- Context awareness, mobility awareness, energy awareness
- Uncertainty

For designing the global architecture, we start from the data end. It seems important to describe the data, and at two different levels: First, the data structure must be available so that the data access service can use mediators to find appropriate mappings between data sources. Second, data itself must be described, for instance for the semantic indexation service or the adaptation service. Semantic data description is maintained by metadata associated to data (in [31], we proposed to index the documents according to their content). We propose to keep these metadata in a database, one database by data source, being on the hosting machine of the data themselves. The accesses on these metadata (availability and access permissions) are those of the data. Moreover, the metadata are generally small compared to the original data, they thus do not increase the load of the data server. The metadata are extracted either manually (given at the same time than the data), or automatically using services available on the P-Grid. If the storage site do not have enough space or the user does not want to store these metadata for any reason, this latter solution allows to reconstruct dynamically the metadata.

One of the consumers of these metadata is the indexation service. This service is replicated on the P-Grid, to balance the load, to make it closer to the data and to increase the robustness of the system. One indexation service is responsible for a set of data sources, each service collaborating with others to have a larger view of the available data, wrt the limit of local index size. The indexing of data in the related indexation service can be either automatic a priori (when the data enters the data source, they register with the indexation service), or a posteriori. In this latter approach, developed in [26], the indexing is done when data is transmitted on the network: Indeed, active routers can examine passing packets and decide on their indexation. Unfortunately, this solution only indexes data being requested, and is complimentary to the a priori solution.

Once the data indexed, the search is facilitated since one must only contact the indexation services. A collaboration between them must be constructed so that they can exchange their content. We have shown in [31,9] that a two-levels collaboration is suitable and is enough, allowing to exchange only a small amount of the data while still being efficient during the search.

The next step is to effectively access the data, and thus to secure this access. We must handle AAA (Authentication, Authorization and Accounting). First, we secure the access to the site itself. We use an authentication mechanism based on a distrust certification model [32]. The idea is, for a given user identified on a home site H to gain access to other sites thanks to a trust chain between these and the original site H. The user gets a home certificate and gain trust certificates during his visits. Each certificate embeds a profile, which is correlated with local permissions. Each device that may accept incoming visitors must host the authentication service of its site. Without any certificate, visitors access limited resources (default profile). Second, data access must be controlled at a fine grain, as close as possible to the data, in order for the SOA of the data to give minimal sufficient rights; the sites holding the data must control the given permission on the fly. We propose to use Sygn [34,36] for this purpose. Sygn allows for a role based access control, and is based on certificate given by the SOA of the data to the potential user of this data. The process involves only these two users and do not need any stable infrastructure. Only an access control service must be deployed where the data are held, which controls the validity of the presented certificates according to the action being requested on the data. This control is only based, from the storage point of view, on the association between the identifier of the SOA of the data and the identifier of the data. One important characteristic is the necessity to have unique identifier of the data on the P-Grid: This can be achieved through the use of users public keys. These two mechanisms (authentication and authorization) use certificates: These are generally small in size and can be embedded on mobile devices with the users (usb keys, ...). This is preferable to a central storage site, being a single point of failure and a privileged target for attacks. Nevertheless, we can imagine a proxy acting for the user as a cache when certificates are needed (for performance and ease of use reasons). Third, concerning the accounting, Sygn can be set so that every accesses to the data are stored together with the data. We believe the access to

the data is the right location for this accounting, providing we store the identifier of the requester and his associated action. Finally, security is also achieved by encrypting data. An encryption service must be provided in order to increase the confidentiality and non disclosure of the data on the devices. This service can be embedded on devices, but some permanent services on the stable infrastructure must exist for those too short in computing resources. We propose to use CryptStore [35], which rely on the distribution of key shares on the network. This flexible solution allows the user to delegate the encryption/decryption service out of its device, while keeping a secure control of the encrypted data. The key shares to decrypt the data are spread on storage sites, where their access is controlled in the same way than the data. One advantage of this solution is the fact that the SOA does not need any key management procedure. Another advantage is the fact that only a parameterizable subset of all the key shares are necessary to decrypt the data, increasing the reliability of the system when some key servers are down or unreachable.

Once the data accessed, it can be interesting to create replicas of these, or more simply to cache them. The replication service presented in [20] needs monitoring services to optimize the placement of the replicas. This monitoring must give up-to-date information on the status of network resources, and aggregate these so that it reflects the cost to get a file. On the other side, the cache service [9] needs the semantic indexation service presented above to adapt the replacement policies to the data usage. It also handles metadata thus it needs a database to keep those.

Before delivering the data to the user, it can be adapted to his context. This adaptation (or any treatment on the data), will use services available on the P-Grid. It needs to interact with a middleware that will launch, execute and monitor the services. Moreover, the adaptation service [6] is based on the description of available services and on the adequate representation of the context.

More generally, a service to discover and register available services is mandatory and should exist on every devices. This local service interacts with other services on the stable infrastructure, and with its surroundings to discover the locally available services.

From the user point of view, only a discovery service and a set of certificates must be present with him. These will allow him (and, by delegation his device) to reconstruct part by part his (its) view of the P-Grid.

5 Implementation Issues

The architecture described in the previous sections is not working as a whole set. Nevertheless, efforts in our group for implementing parts of it have already been done and could be successfully reutilized and integrated in a P-Grid: Some works [6,31,32,20] have been initiated in the context of pervasive systems, others in the context of data grids [35,34,36,9].

In the case of the data grids, we have developed our contributions using standard middlewares like Globus [1], conforming to the OGSA/WSRF

architecture [4]. OGSA offers the desired functionalities of the components of the architecture, the infrastructure being assured by WSRF (Web Service resource Framework). On the data side of the architecture, OGSA does not offer an integrated view by rather a set of tools and services to access the data (OGSA-DAI-Data Access and Integration), to handle the replicas (RMS - Replica Management Service), to transfer the data (GridFTP). Moreover, some activities that are directly related to data management are studied as different fields, like for instance the information service (that can provide information on data sources, storage resources, ...). Some problems we face using OGSA is the heaviness of the protocols and the architecture. As an example, in WSRF, the service calls are encapsulated in SOAP, that adds a large amount of payload to the service. In a light device infrastructure, the cost of this protocol can be seen as a potential bottleneck. Moreover, it is difficult to separate efficiently the services in OGSA to manage a light architecture, embeddable on light devices with poor memory and network connections. Finally, OGSA does not deal with mobility, locality, context-awareness in its development. But it can be a relatively good starting point ([11],[13]) and since it is widely adopted and standardized. But one would have to adapt the services to pervasive environment, and to work on the effective modularity of the architecture.

On the pervasive systems side, we have developed a software platform called PerSE [33,7] that can host pervasive services. It is user oriented, and is based on the principle of service composition to handle user wills (the users describe partial actions that are transformed in fully described actions based on the context of use). Available services can be constructed outside PerSE, only their interface (messages, protocols) have to be known in PerSE for the composition. The core PerSE base is small, and can be extended using security or log services, independently developed and attached on demand to PerSE. Efficiency and development reactivity is a key concept in the construction of PerSE, thus it does not rely on well established standards. The drawback is the limited easiness to distribute and extend the PerSE base as is.

In our P-Grid approach, we plan to take some ideas from the two worlds, with the flexibility of PerSE to construct services and chain them, and the ambition of OGSA for its wide adoption and standardization process.

6 Conclusion and Future Works

In this article, we first gave our vision of a Pervasive Grid, from the data side point of view. We exhibit existing solutions and show their limitation, particularly for the data management side. We then identified a number of services useful in this context, and we explained their respective roles. We tried to organize these and to describe their mutual links and the links with an underlying infrastructure, as well as some already developed bricks. We illustrated our approach with an use-case before giving some trails for a real implementation.

Pervasive Grid represents a tremendous and exiting environment in which many known solutions from classical distributed systems do not apply well. We

foresee the development of dedicated algorithm and a continuous interest of researchers worldwide, with the current inclusion of more and more devices (known or hidden) in our everyday life.

Acknowledgments

The author would like to thank Lionel Brunie and Manish Parashar for the interesting exchanges on Pervasive Grid issues, as well as all the PhD and Master students that worked in the previous years with me at INSA-Lyon and University Paul Sabatier in Toulouse.

References

1. The globus alliance, `http://www.globus.org`
2. Project Akogrimo (2004), `http://www.akogrimo.org`
3. Allen, G., Davis, K., Dolkas, K.N., Doulamis, N.D., Goodale, T., Kielmann, T., Merzky, A., Nabrzyski, J., Pukacki, J., Radke, T., Russell, M., Seidel, E., Shalf, J., Taylor, I.: Enabling applications on the grid: A Gridlab overview. International Journal of High Performance Computing Applications: Special issue on Grid Computing: Infrastructure and Applications (to appear, 2003)
4. OGSA: Open Grid Services Architecture, `http://www.ggf.org`
5. Baude, F., Caromel, D., Huet, F., Vayssiere, J.: Communicating mobile active objects in java. In: Williams, R., Afsarmanesh, H., Bubak, M., Hertzberger, B. (eds.) HPCN-Europe 2000. LNCS, vol. 1823, pp. 633–643. Springer, Heidelberg (2000)
6. Berhe, G., Brunie, L., Pierson, J.-M.: Content adaptation in distributed multimedia systems. Journal of Digital Information Management, special issue on Distributed Data Management 3(2) (June 2005)
7. Bihler, P., Brunie, L., Scuturici, V.-M.: Modeling user intention in pervasive service environments. In: Yang, L.T., Amamiya, M., Liu, Z., Guo, M., Rammig, F.J. (eds.) EUC 2005. LNCS, vol. 3824, pp. 977–986. Springer, Heidelberg (2005)
8. Bruneo, D., Scarpa, M., Zaia, A., Puliafito, A.: Communication paradigms for mobile grid users. In: CCGrid 2003, p. 669 (2003)
9. Cardenas, Y., Brunie, L., Pierson, J.-M.: Uniform distributed cache service for grid computing. In: Andersen, K.V., Debenham, J., Wagner, R. (eds.) DEXA 2005. LNCS, vol. 3588. Springer, Heidelberg (2005)
10. Cherniack, M., Franklin, M.J., Zdonik, S.B.: Data management for pervasive computing. In: VLDB 2001: Proceedings of the 27th International Conference on Very Large Data Bases, p. 727. Morgan Kaufmann Publishers Inc., San Francisco (2001)
11. Chu, D., Humphrey, M.: Bmobile ogsi.net: Grid computing on mobile devices. In: Grid Computing Workshop (associated with Supercomputing 2004), Pittsburgh, USA (November 2004)
12. Clarke, B., Humphrey, M.: Beyond the "device as portal": Meeting the requirements of wireless and mobile devices in the legion grid computing system. In: 2nd International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing (associated with IPDPS 2002), Ft. Lauderdale, USA (April 2002)

13. Coronato, A., De Pietro, G.: Mipeg: A middleware infrastructure for pervasive grids. Journal of Future Generation Computer Systems (2007)
14. Coulson, G., Grace, P., Blair, G., Duce, D., Cooper, C., Sagar, M.: A middleware approach for pervasive grid environments. In: UK-UbiNet/ UK e-Science Programme Workshop on Ubiquitous Computing and e-Research (April 2005)
15. Davies, N., Friday, A., Storz, O.: Exploring the grid's potential for ubiquitous computing. IEEE Pervasive Computing 3(2), 74–75 (2004)
16. Foster, I., Kesselman, C. (eds.): The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc., San Francisco (1999)
17. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The physiology of the grid: An open grid services architecture for distributed systems integration (2002)
18. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. International Journal of Supercomputing Applications 15(3), 200–222 (2001)
19. Gonzalez-Castano, F.J., Vales-Alonso, J., Livny, M., Costa-Montenegro, E., Anido-Rifo, L.: Condor grid computing from mobile handheld devices. SIGMOBILE Mob. Comput. Commun. Rev. 7(1), 117–126 (2003)
20. Gossa, J., Pierson, J.-M., Brunie, L.: Fredi flexible replicas displacer. In: International Conference in Networking, Mauritius, April 2006. IEEE CS Press, Los Alamitos (2006)
21. Graboswki, P., Lewandowski, B., Russell, M.: Access from j2me-enabled mobile devices to grid services. In: Proceedings of Mobility Conference 2004, Singapore (2004)
22. Hingne, V., Joshi, A., Finin, T., Kargupta, H., Houstis, E.: Towards a pervasive grid. In: International Parallel and Distributed Processing Symposium (IPDPS 2003), p. 207 (2003)
23. Jean, K., Galis, A., Tan, A.: Context-aware grid services: Issues and approaches. In: Bubak, M., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2004. LNCS, vol. 3038, p. 1296. Springer, Heidelberg (2004)
24. Jiang, S., O'Hanlon, P., Kirstein, P.: Moving grid systems into the ipv6 era. In: Li, M., Sun, X.-H., Deng, Q.-n., Ni, J. (eds.) GCC 2003. LNCS, vol. 3033, pp. 490–499. Springer, Heidelberg (2004)
25. Kurkovsky, S., Bhagyavati, Ray, A., Yang, M.: Modeling a grid-based problem solving environment for mobile devices. In: ITCC (2), p. 135. IEEE Computer Society, Los Alamitos (2004)
26. Lefevre, L., Pierson, J.-M., Guebli, S.-A.: Deployment of collaborative web caching with active networks. In: Wakamiya, N., Solarski, M., Sterbenz, J.P.G. (eds.) IWAN 2003. LNCS, vol. 2982, pp. 80–91. Springer, Heidelberg (2004)
27. McKnight, L., Howison, J., Bradner, S.: Wireless grids, distributed resource sharing by mobile, nomadic and fixed devices. IEEE Internet Computing 8(4), 24–31 (2004)
28. Parashar, M., Pierson, J.-M.: When the Grid becomes pervasive: A vision on Pervasive Grids. In: Hellenic European Conference on Computer Mathematics & its Applications (HERCMA), Athens, Greece, 20/09/2007-22/09/2007 (2007)
29. Park, S.-M., Ko, Y.-B., Kim, J.-H.: Disconnected operation service in mobile grid computing. In: Orlowska, M.E., Weerawarana, S., Papazoglou, M.P., Yang, J. (eds.) ICSOC 2003. LNCS, vol. 2910, pp. 499–513. Springer, Heidelberg (2003)
30. Phan, T., Huang, L., Dulan, C.: Challenge: integrating mobile wireless devices into the computational grid. In: MobiCom 2002: Proceedings of the 8th annual international conference on Mobile computing and networking, New York, NY, USA, pp. 271–278. ACM Press, New York (2002)

31. Pierson, J.-M., Brunie, L., Coquil, D.: Semantic collaborative web caching. In: Web Information Systems Engineering 2002 (ACM/IEEE WISE 2002), Singapore, December 2002, pp. 30–39. IEEE CS Press, Los Alamitos (2002)
32. Saadi, R., Pierson, J.-M., Brunie, L.: Apc: Access pass certificate, distrust certification model for large access in pervasive environement. In: Proceedings of International Conference on Pervasive Services IPCS 2005, Santorini, Greece (July 2005)
33. Scuturici, V.-M., Pierson, J.-M., Brunie, L.: Perse: Pervasive services environment, research report (2005)
34. Seitz, L.: Conception et mise en oeuvre de mécanismes sécurisés de partage de données confidentielles: application á la gestion de données biomédicales dans les grilles de calcul. PhD thesis, INSA de Lyon (July 2005)
35. Seitz, L., Pierson, J.-M., Brunie, L.: Encrypted storage of medical data on a grid. Journal Methods of Information in Medicine 44(2), 198–202 (2005)
36. Seitz, L., Pierson, J.-M., Brunie, L.: Sygn: A certificate based access control in grid environments. Technical Report 2005-2007, LIRIS (July 2005)
37. Srinavisan, S.H.: Pervasive wireless grid architecture. In: Second Annual Conference on Wireless On-demand Network Systems and Services, WONS 2005 (2005)
38. Stockinger, H., Donno, F., Laure, E., Muzaffar, S., Kunszt, P., Millar, P.: Grid data management in action: Experience in running and supporting. In: The EU DataGrid Project, in Computing in High Energy Physics (CHEP 2003), La Jolla, pp. 24–28 (2003)
39. Storz, O., Friday, A., Davies, N.: Towards 'ubiquitous' ubiquitous computing: an alliance with 'the grid'. In: First Workshop on System Support for Ubiquitous Computing Workshop (Ubisys 2003) in association with Fifth International Conference on Ubiquitous Computing, Seattle, Washington, U.S (October 2003)
40. Weiser, M.: The computer for the 21st century. Scientific American 265(3), 66–75 (1991)
41. Yamin, A., Augustin, I., Barbosa, J., da Silva, L., Real, R., Cavalheiro, G., Geyer, C.: Towards merging context-aware, mobile and grid computing. International Journal of High Performance Computing Applications 17(2), 191–203 (2003)
42. Zhang, G., Parashar, M.: Dynamic context-aware access control for grid applications. In: 4th International Workshop on Grid Computing (Grid 2003), Phoenix, AZ, USA, pp. 101–108. IEEE Computer Society Press, Los Alamitos (2003)

DTR: Distributed Transaction Routing in a Large Scale Network*

Idrissa Sarr, Hubert Naacke, and Stéphane Gançarski

University Paris 6, LIP6 Lab, France
FirstName.LastName@lip6.fr

Abstract. Grid systems provide access to huge storage and computing resources at large scale. While they have been mainly dedicated to scientific computing for years, grids are now considered as a viable solution for hosting data-intensive applications. To this end, databases are replicated over the grid in order to achieve high availability and fast transaction processing thanks to parallelism. However, achieving both fast and consistent data access on such architectures is challenging at many points. In particular, centralized control is prohibited because of its vulnerability and lack of efficiency at large scale. In this article, we propose a novel solution for the distributed control of transaction routing in a large scale network. We leverage a cluster-oriented routing solution with a fully distributed approach that uses a large scale distributed directory to handle routing metadata. Moreover, we demonstrate the feasibility of our implementation through experimentation: results expose linear scale-up, and transaction routing time is fast enough to make our solution eligible for update intensive applications such as world wide online booking.

1 Introduction

Grid systems use a distributed approach to deal with heterogeneous resources, high autonomy and large-scale distribution. Thus, they present a real interest to important areas of enterprise information systems. For instance, Global Distribution Systems (GDS) like Amadeus [2], Sabre [17], Galileo [6] manage a huge amount of data for airline companies, hotels and travel agencies. For instance, Amadeus system information manages data for 62.000 travel agencies, 734 airline companies, 61560 hotels and covers more than 207 countries. The challenge for these systems is to ensure data availability and consistency in order to deal with fast updates. To solve this problem, these systems use expensive parallel servers. Furthermore data is located on single site, which limits scalability and availability. Mapping these GDS systems to a grid allows to overcome these limitations at a rather low cost. In such architecture, the data accessed by the GDS will be stored by the participants (hotels, airline companies, *etc.*) and can be shared. Thus, the data is distributed and parallel executions can be done so that load balancing is achieved.

* This work has been partially financed by the Respire project (ANR-05-MMSA-0011).

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 521–531, 2008.

In order to improve data availability, data is replicated and transactions are routed to the replicas. However, the mutual consistency can be compromised, because of concurrent updates. Let us illustrate the problem with a concurrent update. Assume that we have two replicas R_1 and R_2 and we have two transactions T_1 and T_2 which are sent respectively by two applications (or travel agency) A_1 and A_2. Each transaction aims for making a reservation operation in the same flight (AF709) of Air France airline company. Assuming that only one seat is available and T_1 are routed to R_1 and T_2 to R_2, then the simultaneous execution of T_1 and T_2 produces a data inconsistency: one of travel agencies sales a non-existing seat. Another point is that some queries can be executed at a node which misses the latest updates. For instance, a request which computes the number of passengers of a flight can be executed in a (few loaded) stale node. To this end, two conditions must be satisfied: *(i)* the staleness of the node, expressed in number of missing updates, does not exceed the quantity of overbooking the company is allowed, and *(ii)* the request does not perform updates, for sake of consistency. In other words, controlling the freshness of nodes for executing read-only queries can help in improving performances through a better load balancing. Many solutions have been proposed in distributed systems for managing replicas [14], [12], [9], [8], [5] and [13]. Some of them include freshness control [16], [7], [10] and [1]. We base our work on the Leg@net approach [7], since it offers update anywhere and freshness control features and does not require any modification of the underlying DBMS nor of the application source code.

However, cluster systems deal with homogeneous nodes and are not suitable for systems which have heterogeneous and independent entities such as GDS. In order to make Leg@net system suitable to GDS applications, it is necessary to modify its architecture such that it becomes fully distributed on grid system. To reach this goal, the router and the metadata will be replicated at many sites of the grid.

In this paper, we aim to design a new system relying on the Leg@net principles to deal with transaction routing at a large-scale. Our main contributions are:

- A fully distributed transaction routing model which targets update-intensive applications. Our middleware, ensures data distribution transparency and does not require synchronization (through 2PC or group communication) while updating data.
- A large-scale distributed directory for metadata, highly available and easy to access. It enables to keep data consistency with few communications messages between routers.
- Experimental evaluation of our approach that show its feasibility.

The rest of this paper is organized as follows. We first present in Section 2 the global system architecture, the replication and freshness model. Section 3 describes our algorithm for transaction routing with freshness control. Section 4 presents experimental evaluations of our system and Section 5 concludes.

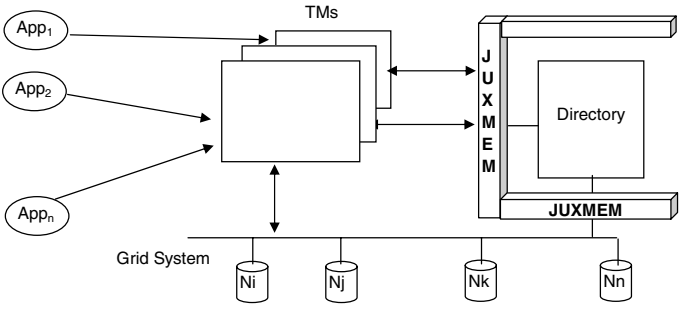


Fig. 1. Global architecture

2 System Architecture and Model

In this section we describe how our system architecture and model are defined. We first present the global architecture which is needed for understanding our solution. Then we describe the replication and freshness model used in order to preserve global consistency.

2.1 Architecture

The global architecture of our system is depicted on Figure 1. Transactions are sent by applications to any Transaction Manager (*TM*). TM uses metadata stored in a shared directory implemented into JuxMem [3], to route the transaction for execution on a data node (N^i) while maintaining global consistency. JuxMem provides the abstraction of a shared memory over a distributed grid infrastructure, by transparently handling consistency in a fault-tolerant way. Data nodes use a local relational DBMS to store data and performs local execution of transactions sent by TMs.

2.2 Replication and Freshness Model

We assume a single database composed of relations $R^1, R^2...R^n$ that is fully replicated at nodes $N_1, N_2...N_m$. The local copy of R^i at node N_j is denoted by R^i_j and is managed by the local DBMS. We use a lazy multi-master (or update everywhere) replication scheme. Each node can be updated by any incoming transaction and is called the initial node of the transaction. Other nodes are later refreshed by propagating the transaction through refresh transactions. We distinguish between three kinds of transactions:

- Update transactions are composed of one or several SQL statements which update the database.
- Refresh transactions are used to propagate update transactions to the other nodes for refreshment. They can be seen as "replaying" an update transaction

on another node than the initial one. Refresh transactions ar distinguished from update transactions by memorizing in the shared directory, for each data node, the transactions already routed to that node.
- Queries are read-only transactions, and thus need not be refreshed.

Let us note that, because we assume a single replicated database, we do not need to deal with distributed transactions, *i.e.* , each incoming transaction can be entirely executed at a single node.

2.3 Freshness Model and Metadata

Every transaction (update, refresh or query) reads a set of relations, every update and refresh transaction writes a set of relation. This information can be obtained by parsing transactions code, and is stored into the shared directory.

Queries may access to stale data, provided it is controlled by applications. To this end, application can associate a *tolerated staleness* with queries. Staleness can be defined through various measures [10]. In this paper, we only consider one measure, defined as the number of updated tuples, for each relation R_i accessed by a transaction T. More precisely, the staleness of R_j^i is equal to the maximum number of tuples of R^i already updated on any node but not yet updated on N_j. The tolerated staleness of a query is thus, for each relation read-accessed by the query, the maximum number of updates that can be missing on a node to be read by the query. Tolerated staleness reflects the freshness level a query requires to be executed on a given node. For instance, if the query requires perfectly fresh data, its tolerated staleness is equal to zero. This information is also stored in the shared directory. Note that, for consistency reasons, update (and thus refresh) transactions must read perfectly fresh data, thus their tolerated staleness is always equal to zero for every relation they access.

To compute the staleness of a relation copy R_j^i, we store in the shared directory, for each update transaction T writing R^i, the maximum number of tuples T may update on R^i. We also store the system global state, *i.e.* for each update transaction, the nodes where it has been already executed. This allows for computing a lower bound of R_j^i's staleness, which is lower or equal to the actual staleness. This guarantees that, when executing a query with tolerated staleness ts on a node with an estimated staleness $s \leq ts$, then the actual freshness of the node is sufficient to fulfil the query requirement.

The shared directory also stores, for each transaction T, the estimated time of processing T, which is a moving average based on previous executions of T. It is initialized by a default value obtained by running T on an unloaded node. It serves at computing the cost function used for transaction routing and load balancing (see Section 3.1).

2.4 Global Consistency

In a lazy multi-master replicated database, the mutual consistency of the database can be compromised by conflicting transactions executing at different nodes. To solve this problem, update transactions are executed at database

nodes in compatible orders, thus producing mutually consistent states on all database replicas. Queries are sent to any node that is fresh enough with respect to the query requirement. This implies that a query can read different database states according to the node it is sent to. However, since queries are not distributed, they always read a consistent (though stale) state. To achieve global consistency, we maintain a graph in the shared directory, called global precedence order graph. It keeps track of the conflict dependencies among active transactions, *i.e.* , the transactions currently running in the system but not yet committed. It is based on the notion of potential conflict: an incoming transaction potentially conflicts with a running transaction if they potentially access at least one relation in common, and at least one of the transactions performs a write on that relation. This pre-ordering strategy, already used in Leg@net, is comparable to the one of [4]. The main difference is that the global ordering graph is also used for computing nodes freshness

3 Transaction Routing with Freshness Control

In this section, we describe how transactions are routed in order to improve performance. First, we present the routing algorithm, directly inspired from [7]. Then, we discuss the specific issues raised by the use of a shared directory.

3.1 Routing Algorithm

Our routing strategy is cost based and uses late synchronization, thus it takes into account the cost of refreshing a node before sending a transaction T to it. As mentioned in [7], the routing complexity is linear in the number of active transactions and the number of nodes, which makes our approach scalable. The cost-based routing algorithm evaluates, for each node N_j:

- N_j's load. This cost is computed by evaluating the remaining execution time of all running or waiting transactions at node N_j.
- the cost of refreshing N_j enough (if necessary) to meet the freshness requirement of T. To this end, it computes a refresh sequence S for N_j: the minimal sequence of refresh transactions to be executed on N_j to make it fresh enough *wrt.* T's requirement. In other words, after applying the refresh sequence on N_j, its staleness *wrt.* each relation read-accessed T is lower than the respective staleness tolerated by T (remember that this tolerated staleness is always 0 if T is an update). The cost is the estimated time needed to execute the sequence S.
- the cost of executing T itself.

Then it chooses the node N which minimizes the cost, *i.e.* the sum of the preceding three costs, and sends to N the sequence S followed by T. It also updates the shared directory: all the transactions in S (plus T if T is an update) are dropped from the set of transactions waiting to be executed on N. In order to ensure global consistency, refresh transactions are inserted in the refresh sequence according

to the global serialization order: whenever a refresh transaction is inserted, all its predecessors not yet executed on the node are also inserted, in the appropriate order, so that the sequence order is compatible with the global precedence order (see Section 2.4)

3.2 Concurrent Access to the Shared Directory

As opposed to the centralized version of [7], where a single router is interacting sequentially with the directory, we must here take into account the concurrency problem due to the presence of several routers, thus to simultaneous access the metadata. We decided to solve this problem using traditional two phase locking (locks on metadata are kept until the end of the routing process), based on two observations: (1) the routing process is very fast compared to the execution of the refresh sequence and of the transaction itself, thus locks are released rather quickly, and (2), locking is provided by JuxMem, which makes the implementation straightforward. In order to validate this choice, we ran experiments to measure the overhead due to concurrent access to the shared directory (see next Section).

4 Experimental Validation

In this section we evaluate the performances of our solution through experimentation. In [7], the Leg@net router was demonstrated to perform better than well known routing strategies such as round robin or least loaded node routing. Since our solution relies on the same cost based routing algorithm, we focus here on comparing the distributed version of the routing algorithm with the Leg@net centralized one.

The experiments follow two goals. First, we need to check that the distributed router is not a bottleneck, *i.e.* , it routes every transaction fast enough. Second, we want to assess if the distributed router brings some global benefit for the applications *i.e.* , if it improves transaction response time.

4.1 Experimental Setup

We run all the experiments on a 20 nodes (P4, 3GHz, 2GB RAM) cluster with 1Gb/s inter node connection as well as some desktop computers from the laboratory to host end user applications. The router is implemented in C language and relies upon JuxMem services, which are built on top of Sun JXTA layer. JuxMem provides a grid-wide RAM access. Our router acts as a middleware; it provides a transaction processing interface for the applications. A cluster node has two roles: it acts as a router node and/or a DBMS node.

4.2 Distributed Directory Access Overhead

The first set of experiments focuses on the routing step itself. It measures the overhead of using a distributed directory to manage router metadata. The

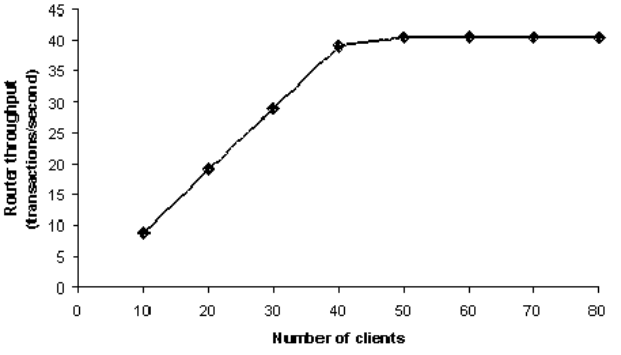


Fig. 2. Middleware throughput

workload is made of an increasing number of applications, each of them is sending one transaction per second to a single router. We measure the resulting throughput (in transactions/second) that the router achieves. Figure 2 shows that a single router can process up to 40 transactions per second. This threshold is satisfying considering that more routers would be able to handle higher workloads.

A part of the routing process is to access the distributed directory. In order to quantify the directory access overhead and then to know if our approach can scale out, we increase the directory size by adding database replicas, since more replicas imply more metadata. We report on Figure 3, the output workload that a router achieves in 3 cases: small , medium and large directory size (respectively 5 , 50, 100 replicas). We measure a slowdown of less than 20% for a large directory that has a replication degree of 100. For a smaller replication degree of 50, the slow down is only 5%. Since most of the applications, in our context, require a replication degree lesser than 10, we conclude that the distributed directory access is not a performance brake.

Furthermore, we study the impact of multiple routers concurrently accessing the distributed directory. The workload is made of the same applications as the former experiment, but the transactions are sent to 2 routers (such that half of the workload goes to each router). The results of Figure 4 are obtained in the worst case (*i.e.* all transactions access to the same data leading the routers to do so with metadata) and they shows a maximal throughput of 20 transactions/second that is half of the standalone throughput. Indeed, waiting for locks is decreasing the router throughput. Thus, in the worst case where each directory access is delayed by a concurrent access to the same metadata, the router is still able to provide reasonable throughput. We note that, in our context, concurrent situations are not frequent since metadata is fragmented and the probability of concurrent access to the same fragment is weak. Nevertheless, ongoing experimentations aim to evaluate precisely the slowdown led by concurrent access to the distributed directory *wrt* concurrency degree. In other words, we will vary the concurrency degree between 0% and 100% and measure the variations of the performances.

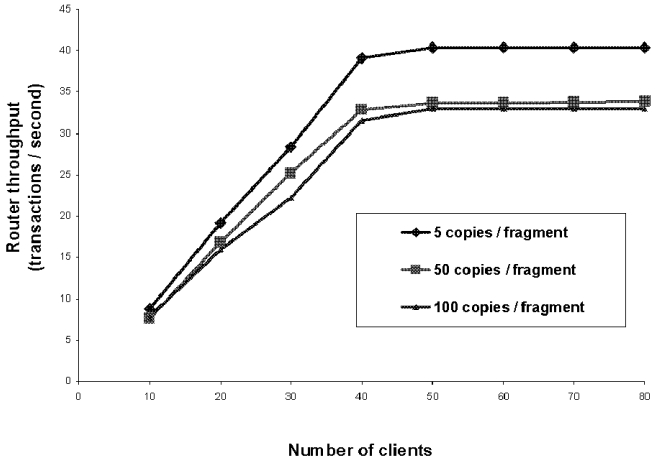


Fig. 3. Directory size overhead

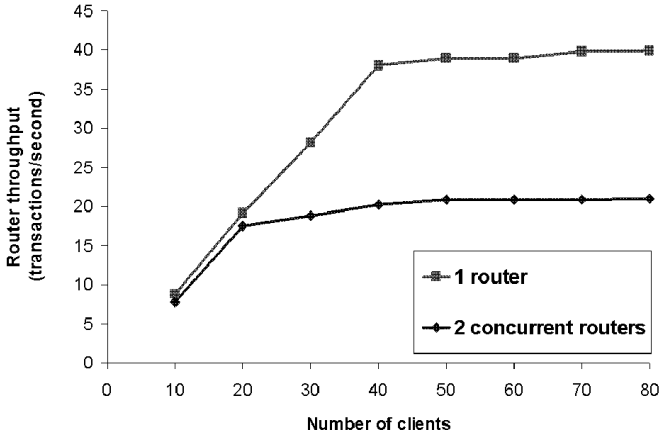


Fig. 4. Concurrent access

4.3 Overall Routing Performance

This experiment focuses on the overall transactional performance of the distributed routing (DR). We measure the increase in throughput compared to the centralized routing (CR) of [7]. The workload is made of N applications of 3 kinds (N/3 apps of each kind). Each kind of application is accessing a distinct part of the database and is connected to a distinct router. In other words, there is no concurrency between routers when accessing the directory We measure the output throughput when N is varying from 15 to 150 applications. On Figure 5, we compare theses results with a case where a single router receives the whole identical workload. As n is increasing, the gap between DR and CR is expand-

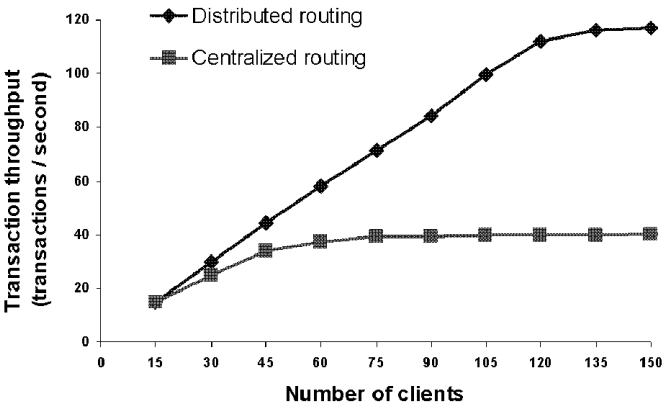


Fig. 5. Distributed vs centralized algorithm

ing. For a heavy workload of 150 applications, DR outperforms CR by a ratio of 3. The main reason is that centralized routing quickly reaches its performance limit due to the time required to route each transaction. We note that the benefit ratio equals the number of routers: that demonstrates a linear scale up. Ongoing experimentations are conducted to assess up to which number of routers our solution scales linearly.

4.4 Dealing with Scale Up

In order to deal with a large scale network, our experiments must take into account the databases and directory replication at large scale such as grid systems. However, in this paper, our main goal is to demonstrate the performance benefit that we achieve by distributing the routing protocol. To this end, replicating our middleware over few nodes, at least one router per cluster, is sufficient. More precisely, we note that JuxMem experiments [11] reported the time to write metadata stored on a remote cluster that belongs to the Grid5000 [15] infrastructure: more than 90 milliseconds per write. In such an environment, the routing time of our system would be around 100 milliseconds. Then, the router throughput falls to 10 transactions per second.

However, if every router access only a part of the distributed directory that is managed locally on the same cluster, the throughput performance (up to 40 transactions/second) is still observed, even if the databases are replicated over many remote clusters. In this case, the response time slightly increases depending on network latency between clusters.

5 Conclusion

This paper presents an ongoing work towards the design and implementation of a grid-based large scale data management system. This system extends Leg@net, a

previous work designed for clusters, to the grid context. It uses JuxMem, a shared main memory system designed for grids, to implement a shared distributed directory which stores metadata useful for transaction routing and freshness control. The experimental evaluations, led on a first version of the system, show that the overhead due to accessing to the distributed directory is rather low. They also show that, using the distributed directory, we can implement several instances of the router in the network. For heavy workloads, this increases significantly the global throughput of the system with respect to the centralized version of the router used in Leg@net.

Ongoing experimentations are conducted to find out the optimal number of router instances with respect to the heaviness of the workload. After, we plan to evaluate precisely the slowdown led by multiple routers concurrently accessing the distributed directory *wrt* to concurrency degree. We will also take into account the grid heterogeneity (intra-cluster links faster than inter-cluster links) in our cost estimations, in order to improve node choice and thus to yield better performances. Next, we will run the same experimentation as the one led with Leg@net to measure the benefit, in terms of response time, that the queries will achieve with respect to the staleness they tolerate. Furthermore, we plan to deal with fault tolerance.

References

1. Akal, F., Türker, C., Schek, H., Breitbart, Y., Grabs, T., Veen, L.: Fine-Grained Replication and Scheduling with Freshness and Correctness Guarantees. In: Int. Conf. on Very Large DataBases, VLDB (2005)
2. Amadeus, `http://www.amadeus.com/index.jsp`
3. Antoniu, G., Bougé, L., Jan, M.: JuxMem: An Adaptive Supportive Platform for Data Sharing on the Grid. Scalable Computing: Practice and Experience 6(3) (2005)
4. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading (1987)
5. Choi, S., Baik, M., Gil, J., Park, C., Jung, S., Hwang, C.: Group-Based Dynamic Computational Replication Mechanism in Peer-to-Peer Grid Computing. In: IEEE Int. Symposium on Cluster Computing and the Grid, CCGrid (2006)
6. Galileo, `http://www.galileo.com`
7. Gançarski, S., Naacke, H., Pacitti, E., Valduriez, P.: The Leganet System: Freshness-aware Transaction Routing in a Database Cluster. Information Systems Journal 32(2) (2006)
8. Guerraoui, R., Schiper, A.: Software-Based Replication for Fault Tolerance. IEEE Computer 30(40) (1997)
9. Koo, R., Toueg, S.: Checkpointing and rollback-recovery for distributed systems. IEEE Transactions on Software Engineering 13(1) (1987)
10. Le Pape, C., Gançarski, S., Valduriez, P.: Refresco: Improving Query Performance Through Freshness Control in a Database Cluster. In: Int. Conf. On Cooperative Information Systems, CoopIS (2004)
11. Monnet, S.: Gestion des Données dans les Grilles de Calcul: Support pour la Tolérance aux Fautes et la Cohérence des Données. Thése de doctorat, Université de Rennes 1 (2006)

12. Pacitti, E., Coulon, C., Valduriez, P., Özsu, T.: Preventive Replication in a Database Cluster. Distributed and Parallel Databases 18(2) (2005)
13. Pacitti, E., Minet, P., Simon, E.: Fast Algorithms for Maintaining Replica Consistency in Lazy Master Replicated Databases. In: Int. Conf. on Very Large DataBases, VLDB (1999)
14. Patino-Martinez, M., Jimenez-Peres, R., Kemme, B., Alonso, G.: MIDDLE-R, Consistent Database Replication at the Middleware Level. ACM Transactions on Computer Systems 28(4) (2005)
15. Grid 5000 Project, http://www.grid5000.org
16. Rohm, U., Bohm, K., Sheck, H., Schuldt, H.: FAS - a Freshness-Sensitive Coordination Middleware for OLAP Components. In: Int. Conf. on Very Large DataBases, VLDB (2002)
17. Sabre, http://www.sabre.com/

Distributed Management of Massive Data: An Efficient Fine-Grain Data Access Scheme

Bogdan Nicolae[1], Gabriel Antoniu[2], and Luc Bougé[3]

[1] University of Rennes 1/IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
bogdan.nicolae@inria.fr
[2] INRIA/IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
gabriel.antoniu@inria.fr
[3] ENS Cachan Brittany/IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
luc.bouge@bretagne.ens-cachan.fr

Abstract. This paper addresses the problem of efficiently storing and accessing massive data blocks in a large-scale distributed environment, while providing efficient fine-grain access to data subsets. This issue is crucial in the context of applications in the field of databases, data mining and multimedia. We propose a data sharing service based on distributed, RAM-based storage of data, while leveraging a DHT-based, natively parallel metadata management scheme. As opposed to the most commonly used grid storage infrastructures that provide mechanisms for *explicit* data localization and transfer, we provide a *transparent* access model, where data are accessed through global identifiers. Our proposal has been validated through a prototype implementation whose preliminary evaluation on the Grid'5000 testbed provides promising results.

1 Introduction

Managing data at a large scale is paramount nowadays. Governmental and commercial statistics, climate modeling, cosmology, genetics, bio-informatics, etc. are just a few examples of fields routinely generating huge amounts of data. It becomes crucial to efficiently manipulate these data, which must be shared at the global scale. In such a context, one important goal is to provide mechanisms allowing to manage massive data blocks (e.g., of several terabytes), while providing efficient fine-grain access to small parts of the data. Several types of applications exhibit such a need for efficient scaling to huge data sizes: databases ([1,2,3]), data mining [4], multimedia [5], etc.

Towards transparent management of data on the grid. The management of massive data blocks naturally requires the use of data fragmentation and of distributed storage. Grid infrastructures, typically built by aggregating distributed resources that may belong to different administration domains, provide an appropriate solution. When considering the existing approaches to grid data management, we can notice that most of them heavily rely on *explicit* data localization and on *explicit* transfers of large amounts of data across the distributed architecture: GridFTP [6], Reptor [7], Optor [7], LDR [8],Chirp [9], IBP [10], NeST [11],

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 532–543, 2008.

etc. Managing huge amounts of data in such an explicit way at a very large scale makes the design of grid application much more complex. One key issue to be addressed is therefore the *transparency* with respect to data localization and data movements. Such a transparency is highly suitable, as it liberates the user from the need to handle data localization and transfers.

However, a few grid data management systems acknowledge that providing a transparent data access model is important by integrating this idea at the early stages of their design. *Grid file systems*, for instance, provide a familiar, file-oriented API allowing to transparently access physically distributed data through globally unique, logical file paths. The applications simply open and access such files as if they were stored on a local file system. A very large distributed storage space is thus made available to those existing applications that usually use file storage, with no need for modifications. This approach has been taken by a few projects like GFarm [12], GridNFS [13], LegionFS [14], etc.

On the other hand, the transparent data access model is equally defended by the concept of *grid data-sharing service* [15], illustrated by the JuxMem platform [16]. Such a service provides the grid applications with the abstraction of a globally shared memory, in which data can be easily stored and accessed through global identifiers. To meet this goal, the design of JuxMem leverages the strengths of several building blocks: consistency protocols inspired by DSM systems; algorithms for fault-tolerant distributed systems; protocols for scalability and volatility support from peer-to-peer (P2P) systems. Note that such a system is fundamentally different from traditional DSM systems (such as TreadMarks, etc.). First, it targets a larger scale through hierarchical consistency protocols suitable for an efficient exploitation of grids made of a federation of clusters. Second, it addresses from the very beginning the problem of resource volatility due to failures or to the lack of resource availability.

Compared to the grid file system approach, this approach improves *access efficiency* by totally relying on main memory storage. Besides the fact that a main memory access is more efficient than a disk access, the system can leverage locality-optimization schemes developed for the DSM consistency protocols.

Limitations. However, the JuxMem grid data-sharing service suffers from some limitations with respect to the efficient storage and access of massive data blocks. Actually, data are not fragmented in JuxMem: each individual data is physically stored as a single data block in the main memory of a storage provider, and possibly replicated as such on multiple backup providers. Consequently, the largest data block that the service is able to store is limited by the size of the RAM of a *single* provider, typically, a few gigabytes. This lack of fragmentation has another drawback regarding load balancing as all accesses to *different* parts of the same massive block are served by the *same* RAM provider.

Recently, the efficient allocation and access of massive data blocks in main memory has been addressed by the JumboMem [17] system. This system is designed for clusters, not for grids. It allows users to manipulate large contiguous data blocks (of the order of 1 TB) using the aggregated RAM of a set of nodes interconnected through a high-speed Infiniband System Area Network. However,

JumboMem is targeted for a *single* user and does not enable data sharing: it does not provide synchronization, nor replication, nor optimized mechanisms for distributed access by *multiple* users. In contrast, a lot of applications in the field of databases and data-mining target *multi-user* environments. This requires adding an efficient concurrency control, which is not natively provided by JumboMem.

Our approach. Our contribution is twofold. First, we propose a data sharing service allowing to store *massive* blocks of data in a distributed, multi-user environment. Second, *efficient fine-grain access* to the data is provided thanks to distributed, RAM-based storage of data fragments, while leveraging a DHT-based metadata management scheme, which is natively parallel.

This paper is organized as follows. Section 2 gives an overview of our architecture and describes how data access operations are handled. Section 3 provides a few implementation details and reports on a preliminary experimental evaluation. Finally, on-going and future work is discussed in Section 4.

2 Enabling Efficient Fine-Grain Access

Our goal is to provide efficient fine-grain access to massive data blocks stored in large-scale distributed environments such as grids. To goal is addressed in the following way. Data is fragmented into small equally-sized chunks (which will be called *pages* below) and distributed across the local memory of a large number of grid nodes, which act as providers of storage space. This fragmentation allows: 1) to store huge data blocks; and 2) to avoid contention for disjoint accesses to pages. To each data block, we associate some metadata allowing to identify and localize the pages that belong to that block. In order to avoid contention for metadata access, metadata is structured in a fine-grained manner to be described below, and stored in a distributed hash table (DHT). Finally, efficient large-scale concurrency both for reads and writes is achieved using *versioning*: concurrent writes to the same page can proceed in parallel on multiple versions of that page. Our contribution lies in the adequate combination of these techniques to achieve efficient fine-grained access to massive data.

2.1 Architecture

Our service relies on a set of distributed processes communicating through remote procedure calls (RPCs). In a typical setting, each process is running on a different physical node.

Data providers are responsible for storing and retrieving individual pages in their local RAM.

A versioning manager is responsible for serializing write requests and for directing read requests to the latest version available for reading.

Metadata providers are responsible for storing information about the identity and localization of the individual pages that make up a data block. In

our design, metadata providers are organized as a Distributed Hash Table (DHT). Details are given in Section 2.3.

A provider manager receives and solves the clients' requests for data providers. Available providers must previously register with this entity.

To interact with the service, client processes simply use a client library, to which they pass a list of DHT gateways and the network id (IP address, port) of the versioning manager. The rest of the system is transparent to the clients.

2.2 User Interface

Clients manipulate massive data blocks through a simple API:

```
block_id = alloc(page_size, data_size)
block_version = write(block_id, local_buffer, offset, size)
read(block_id, block_version, local_buffer, offset, size)
```

Massive blocks are identified and accessed through a globally unique id, generated when the block is allocated. The user is able to control the granularity (page_size) and maximal size of the block (data_size). Fine-grain access for reads and writes is enabled through (offset, size) range queries. *Each write generates a new block version.* Read operations may explicitly reference a block version. By default, they return the latest available version.

2.3 Metadata Organization

Metadata serves the purpose of identifying and localizing the pages corresponding to the range (offset, size) specified by read and write operations. Our design aims at favoring fast concurrent accesses to metadata.

When the user allocates data_size bytes for a block, the service actually allocates adjusted_size bytes, where adjusted_size is the smallest power of 2 larger than data_size. We organize metadata as a full binary tree. At each level, the nodes of the tree cover disjoint (offset, size) ranges. The root covers (0, adjusted_size), that is, the whole data block. An intermediate node covering (offset, size) points to its left child covering (offset, size/2), and to its right child covering (offset + size/2, size/2). Leaves cover single pages and point to the page id and to provider holding the page (see Figure 1).

A tree node covering (offset, size) is identified by a *key*, obtained by applying a hashing function on the tuple (block_id, offset, size, block_version). Intermediate tree nodes store the following information: offset, size, left_key and right_key, which are respectively the keys of its left and right child. Leaves (covering single pages) store a page_id and a provider_id. Tree nodes are stored on the metadata providers using a DHT structure using the keys defined above. This approach is inspired by Merkle trees [18], initially developed to handle Lamport's one-time signatures.

By relying on the DHT architecture and by selecting an adequate hashing function, an even distribution of page requests among metadata providers can

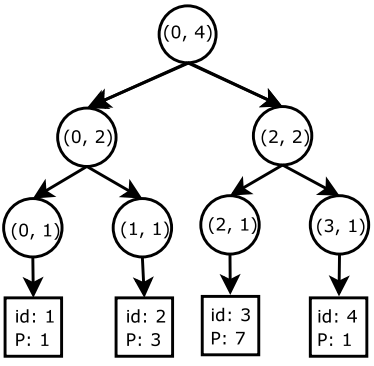


Fig. 1. Metadata representation for a 4-page block. Leaves store page ids Id and the corresponding provider ids P. All nodes are labeled with the (offset , size) range they cover.

be guaranteed with a high probability. Each client takes profit of this even distribution by simultaneously contacting a large number of different gateways to the DHT service when executing parallel requests.

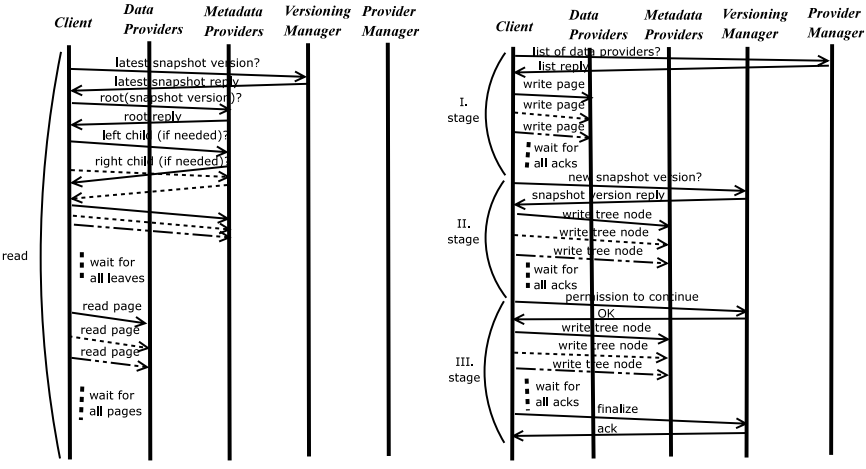
2.4 Managing Allocs, Reads and Writes

Allocation is the cheapest and simplest operation. The client merely contacts the versioning manager providing a page size and total block size. The versioning manager assigns this block an initial version number, 0.

To perform a read if no block version is specified, the client (Figure 2(a)) contacts the versioning manager and requests the latest block version available. If a block version is specified by the read operation, then this step is simply skipped. Then, the client contacts the metadata providers and recursively queries the tree nodes covering the range given by (offset , size) for that particular block version, starting from the root and descending towards the leaves. When a leaf is reached, the client directly contacts the appointed data provider and downloads the actual page. The read operation completes successfully when all the pages have been downloaded. It fails if a node or a page could not be retrieved. In order to enhance parallelism, requests and responses for tree nodes and for pages are handled asynchronously by multiple threads on the client side, and are served in parallel by the various metadata and data providers, respectively.

A write operation (Figure 2(b)) initiated by the client completes in several stages.

1. The client contacts the provider manager to retrieve a list of active data providers available to store the pages in (offset , size) to be written. After receiving the reply, it associates a random data provider and a random page id to each page, so as to uniquely identify the page in the system with high probability. Then, it contacts the data providers, requesting them to store the pages. As in the case of the read operation, write requests sent to



(a) Reading a block: sequence of RPC calls

(b) Writes: sequence of RPC calls

Fig. 2. Managing reads and writes: different line styles denote RPCs running in parallel

providers are asynchronously handled by the client, and served in parallel by the data providers.

2. After all providers acknowledge that the pages have been stored, the client contacts the versioning manager to receive a new version number which shall identify the new block version. The versioning manager enqueues this write request, marks it as pending and returns the version number to the client. After receiving it, the client generates the corresponding tree nodes with respect to the new block version, starting from the leaves up to this new root. All tree nodes whose range is totally included in the interval $[\,\text{offset}\,,\,\text{offset} + \text{size}\,]$ are written to the metadata providers. The rest of the nodes are stored for later processing. The goal of this processing is to properly handle concurrent metadata updates for a single block.
3. Then, the client contacts the versioning manager requesting permission to complete the write operation. If this write request is the oldest one in the queue, then the versioning manager grants permission to complete the write. Otherwise it waits for previous pending writes to be dequeued before granting permission. After receiving permission, the client builds the remaining tree nodes. These nodes cover ranges not included in $[\,\text{offset}\,,\,\text{offset} + \text{size}\,]$. They must correctly reference their children corresponding to nodes not modified by the current write, by using the latest block version previously completed. At the end of this stage, all generated tree nodes are sent to the metadata providers.
4. Finally, the client confirms write completion to the versioning manager, which dequeues the write and marks its corresponding version as the latest block version.

Note that various versions of the same page may be stored on different providers: for each new page version to be written, the least loaded known provider is chosen for its storage, in order to preserve a global load balance in terms of amount of data stored by the providers. (The precise description of this scheme is out of the scope of this paper.)

An important consequence of this property is that successive incremental versions of a data block can be stored as long as storage space is still *globally* available in the system: thanks to our choice of preserving a global load balance, a provider will run out of storage space only when *all* the providers collectively reach their storage limits. In this case, ad-hoc garbage collection can be used to remove the oldest version of the data block. Such a feature has not been implemented in our system, yet.

3 Implementation and Experimental Evaluation

Evaluations are performed using the Grid'5000 [19] testbed, a reconfigurable, controllable and monitorable experimental Grid platform spread over 9 sites geographically distributed in France. We use 160 nodes of a Grid'5000 cluster. Each node has a Intel Pentium 4 CPU running at 2.6 GHz under Linux 2.6 (Ubuntu), outfitted with 4 GB of RAM each, and interconnected by a Gigabit Ethernet network. The theoretical maximum network bandwidth is thus 125 MB/s. However, if we consider the IP and TCP header overhead, this maximum becomes slightly lower: 117.5 MB/s for a MTU of 1500 B. In practice, we could measure 111 MB/s for a standard TCP socket end-to-end transfer.

3.1 Implementation Details

We use BambooDHT [20], which provides a stable, scalable DHT implementation on top of which we build the abstraction of our metadata providers and of the provider manager.

The providers and the versioning manager are implemented in C++ using the Boost C++ collection of libraries. We chose Boost for its standardization throughout the C++ community, and for the wide range of functionalities it provides, among which serialization, threading and asynchronous I/O are of particular interest to us.

3.2 Performance and Evaluation

In this section, we assess the effectiveness of our implementation by running a set of experiments. To support our claim of efficiently dealing with both massive blocks and fine-grain access, we fix the allocated block size at 1 TB, and the page size at 64 kB for all our experiments. Thus, the metadata tree generates a significant overhead as the actual data accesses will concern various continuous ranges from 16 MB up to 1 GB within the overall range of 1 TB.

Using a single client. Our first series of experiments (Figure 3) assesses the overhead of metadata management. We deploy one versioning manager, 100 data providers, and a variable number of metadata providers. Each physical node runs at most one data provider and one metadata provider. A single client writes a series of data, and then reads them back.

It first writes a range size of 16 MB starting from offset 0. Then, it continues writing a second range of 32 MB, starting from the end of the previous range, and so on, doubling the size parameter each time until writing a range of 1 GB. Then, the client successively reads back each of the consecutive segments.

The individual writing and reading times for each segment are logged, sorting out the time used in managing the metadata with respect to the total writing or reading time. Such a cycle is repeated 100 times. This experiment is done for several numbers of metadata providers, that is, several sizes of the DHT, ranging from 5 to 100.

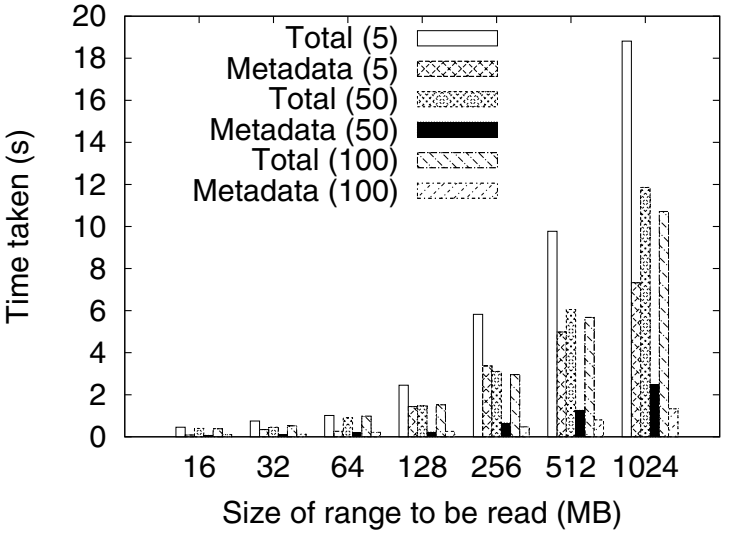
The average timings are reported on Figure 3. Of course, the larger the DHT, the larger the degree of parallelism in accessing its nodes from the client's threads, whence the shorter the overall time.

These timings show an overhead of 18 % for metadata read operations (Figure 3(a)) and 23 % for metadata write operations (Figure 3(b)) for 100 metadata providers. This effectively results in a bandwidth of 92 MB/s for reads and 86 MB/s for writes in accessing the final 1 GB range, to be compared to the maximal limit of 111 MB/s measured in standard TCP socket end-to-end transfer. On the other hand, using only 5 metadata providers results in a metadata management overhead exceeding 68%, which demonstrates the benefits of using a large number of metadata providers, that is, a large DHT.

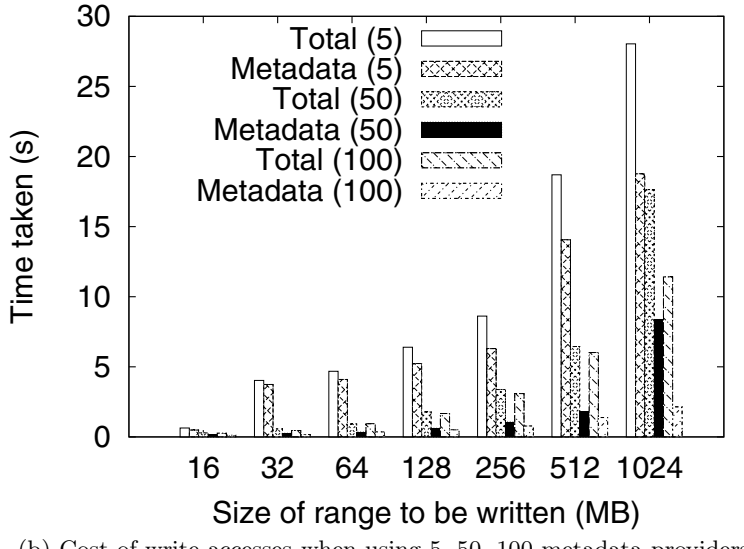
Using multiple concurrent clients. Our next series of experiments (Figure 4) benchmarks our system in a highly concurrent environment, evaluating its scalability when increasing the number of simultaneous reads and writes. For comparison, we also report on what we call an "ideal" bandwidth corresponding to the aggregation of totally independent read (resp., write) operations. That is, we multiply the bandwidth of a single reader (resp., writer) by the number of readers (resp., writers).

We deploy 80 data providers and 80 metadata providers. Each physical node runs one data provider and one metadata provider. The versioning manager is run on a separate node. Then, we deploy a variable number of clients, each of which being run on a separate node, different from the ones used for data, metadata and versioning manager. Clients are synchronized to start simultaneously. They either read or write a disjoint range of the block: client i uses $\text{offset} = i \times 64$ MB, $\text{size} = 64$ MB. For reads, data is prewritten. We measure the average aggregated bandwidth, both for reads and writes, and compare it to the ideal aggregation of bandwidth obtained from a single reader/writer.

As it can be observed, the fine-grained dispersion of data and metadata allows for high bandwidth under heavy concurrency, especially for reads (Figure 4(a)). Writes suffer from a slight performance penalty because of metadata synchronization (Section 2.4). Contacting different providers and different metadata

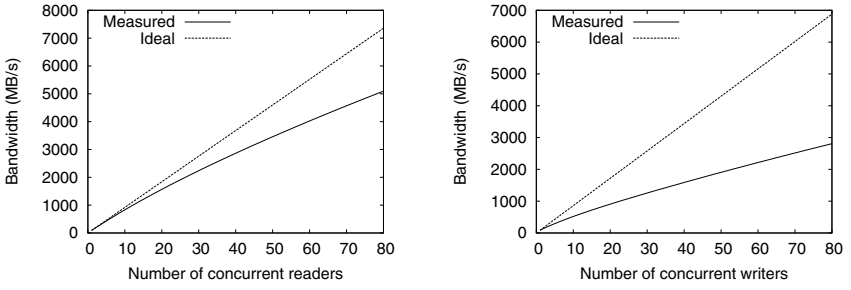


(a) Cost of read accesses when using 5, 50, 100 metadata providers



(b) Cost of write accesses when using 5, 50, 100 metadata providers

Fig. 3. Average data access cost for various contiguous segment sizes and a variable number of metadata providers. For each number of metadata providers, we sort out the time needed to manage the metadata.



(a) Bandwidth for reads (b) Bandwidth for writes

Fig. 4. Average aggregated bandwidth when varying the number of concurrent clients

providers concurrently enables a high degree of load balancing among the network nodes. As such, it makes up for the metadata overhead observed in the first series of experiments.

4 Conclusion

We have addressed the problem of efficiently storing massive data of the order of terabytes in a grid distributed environment. Our contribution consists in proposing a data-sharing service allowing to efficiently allocate, access and modify such massive blocks of data in a distributed, multi-user environment. Efficient fine-grain access to arbitrarily small parts of the data is provided thanks to distributed, RAM-based storage of data fragments, while leveraging a DHT-based, natively parallel metadata management scheme. Preliminary experiments performed with our prototype using the Grid'5000 testbed show that our approach scales well, both in terms of storage providers and in terms of concurrency degree.

Our prototype is however a work in progress and definitely demands further refinement. Fault tolerance, which becomes critical in grid environments, is only partially addressed. We currently leverage some fault-tolerance mechanisms provided by the DHT on which we rely for the implementation of some of the entities of our architecture, the metadata providers and the provider manager. This enhances the availability of metadata thanks to the underlying replication used by the DHT. However, the versioning manager, though not under heavy load, is still a single point of failure in this preliminary scheme. Besides, data is not replicated: for each page, a single copy is kept on a single provider. In order to improve fault-tolerance, replication-based mechanisms could be envisioned in both cases. To this purpose, we intend to explore the possibility to use self-organizing groups to represent these entities, built on fault-tolerant distributed algorithms for atomic multicast, as in [21].

While targeting database, data-mining and multimedia applications, we have not experimented, yet, with a standard implementation that could use our service. We are considering interfacing our service with the PostgreSQL DBMS, in order to provide an efficient support for snapshot isolation.

References

1. Douglas, K., Douglas, S.: PostgreSQL. New Riders Publishing, Thousand Oaks (2003)
2. Thomasian, A.: Concurrency control: methods, performance, and analysis. ACM Computing Survey 30(1), 70–119 (1998)
3. Nicola, M., Jarke, M.: Performance modeling of distributed and replicated databases. IEEE Trans. on Knowl. and Data Eng. 12(4), 645–672 (2000)
4. Jin, R., Yang, G.: Shared memory parallelization of data mining algorithms: Techniques, programming interface, and performance. IEEE Trans. on Knowl. and Data Eng. 17(1), 71–89 (2005)
5. Casey, M.A., Kurth, F.: Large data methods for multimedia. In: Proc. 15th Intl. Conf. on Multimedia (Multimedia 2007), pp. 6–7. ACM, New York (2007)
6. Allcock, B., Bester, J., Bresnahan, J., Chervenak, A.L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnel, D., Tuecke, S.: Data management and transfer in high-performance computational grid environments. Parallel Comput. 28(5), 749–771 (2002)
7. Kunszt, P.Z., Laure, E., Stockinger, H., Stockinger, K.: File-based replica management. Future Generation Computing Systems 21(1), 115–123 (2005)
8. Lightweight data replicator, `http://www.lsc-group.phys.uwm.edu/LDR/`
9. Chirp protocol specification, `http://www.cs.wisc.edu/condor/chirp/`
10. Bassi, A., Beck, M., Fagg, G., Moore, T., Plank, J.S., Swany, M., Wolski, R.: The Internet Backplane Protocol: A study in resource sharing. In: Proc. 2nd IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (CCGRID 2002), Washington, DC, USA, p. 194. IEEE Computer Society, Los Alamitos (2002)
11. Bent, J., Venkataramani, V., LeRoy, N., Roy, A., Stanley, J., Arpaci-Dusseau, A., Arpaci-Dusseau, R., Livny, M.: Flexibility, manageability, and performance in a grid storage appliance. In: Proc. 11th IEEE Symposium on High Performance Distributed Computing, HPDC 11 (2002)
12. Tatebe, O., Morita, Y., Matsuoka, S., Soda, N., Sekiguchi, S.: Grid datafarm architecture for petascale data intensive computing. In: Proc. 2nd IEEE/ACM Intl. Symp. on Cluster Computing and the Grid (Cluster 2002), Washington DC, USA, p. 102. IEEE Computer Society, Los Alamitos (2002)
13. Honeyman, P., Adamson, W.A., McKee, S.: GridNFS: global storage for global collaborations. In: Proc. IEEE Intl. Symp. Global Data Interoperability - Challenges and Technologies, Sardinia, Italy, pp. 111–115. IEEE Computer Society, Los Alamitos (2005)
14. White, B.S., Walker, M., Humphrey, M., Grimshaw, A.S.: LegionFS: a secure and scalable file system supporting cross-domain high-performance applications. In: Proc. 2001 ACM/IEEE Conf. on Supercomputing (SC 2001), pp. 59–59. ACM Press, New York (2001)
15. Antoniu, G., Bertier, M., Caron, E., Desprez, F., Bougé, L., Jan, M., Monnet, S., Sens, P.: GDS: An Architecture Proposal for a grid Data-Sharing Service. In: Future Generation Grids. CoreGRID series, pp. 133–152. Springer, Heidelberg (2006)
16. Antoniu, G., Bougé, L., Jan, M.: JuxMem: An adaptive supportive platform for data sharing on the grid. Scalable Computing: Practice and Experience 6(3), 45–55 (2005)
17. Pakin, S., Johnson, G.: Performance analysis of a user-level memory server. In: IEEE Ann, Intl. Conf. on Cluster Computing (Cluster 2007) (September 2007)

18. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 369–378. Springer, Heidelberg (1988)
19. Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.G., Touche, I.: Grid 5000: A large scale and highly reconfigurable experimental grid testbed. Int. J. High Perform. Comput. Appl. 20(4), 481–494 (2006)
20. Rhea, S., Godfrey, B., Karp, B., Kubiatowicz, J., Ratnasamy, S., Shenker, S., Stoica, I., Yu, H.: OpenDHT: a public DHT service and its uses. In: Proc. 2005 Conf. Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2005), pp. 73–84. ACM, New York (2005)
21. Antoniu, G., Deverge, J.F., Monnet, S.: How to bring together fault tolerance and data consistency to enable grid data sharing. Concurrency and Computation: Practice and Experience 18(13), 1705–1723 (2006)

BLAST Distributed Execution on Partitioned Databases with Primary Fragments*

Daniel Xavier de Sousa[1], Sergio Lifschitz[1], and Patrick Valduriez[2]

[1] PUC-Rio Dep Informatica, Rio de Janeiro - Brazil
{dsousa,sergio}@inf.puc-rio.br
[2] LINA and INRIA, Nantes - France
patrick.valduriez@inria.fr

Abstract. BLAST is one of the most popular computational biology tools. The execution cost of BLAST is highly dependent on database sizes, which have considerably increased following all recent advances in sequencing methods. The evaluation of BLAST in distributed and parallel environments like PC clusters and Grids has been largely investigated in order to obtain better performances. This work evaluates a replicated allocation of the (sequences) database, where each copy is also physically fragmented. We investigate two dynamic workload balancing methods that focus on our database allocation strategy. Preliminary practical results show that we achieve both a balanced workload and very good performances. We briefly discuss ideas that would make our approach feasible for Grid computational environments.

1 Introduction

In this work we are interested in evaluating one of the most popular operations in bioinformatics, namely BLAST - Basic Local Alignment Search Tool - evaluation [2]. BLAST is a popular family of algorithms for (bio)sequences comparison and alignment operations. These operations are widely and often used in laboratories that make Genome sequencing and analysis.

Except for single input queries and small sequence databases, BLAST processing is very time consuming and performance is a key issue regarding that genomic databases are getting bigger. Therefore, many different ideas have been proposed to improve BLAST execution times. Among them, parallel and distributed strategies on top of clusters and grids are available (e.g. [1,8]).

From a database point of view, there are two basic approaches: either the sequence database is fully replicated at all processing nodes (single nodes or clusters), and there is query segmentation, or the database is split into disjoint fragments and the complete query sequences input is executed at all sites [4]. While the replicated-database case is a straightforward inter-query parallelization method, the fragmented situation leads to a more complicated situation:

* Work partially funded by CNPq-INRIA (GriData project).

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 544–554, 2008.

BLAST execution on smaller fragments may not generate the correct (sequential) results if run time statistical parameters (e.g. Z for WU-BLAST [13] and Y for NCBI-BLAST [9]) are not well defined [7].

Furthermore, as in many other parallel computation problems, when there is an uneven workload, all the benefits that come with these approaches may be lost. Therefore, load balancing strategies must be considered [5]. In this paper we propose an approach to execute BLAST on distributed and autonomous environments, where we are mainly concerned with workload balancing. This work evaluates a replicated allocation of the (sequences) database, where each copy is also physically fragmented. We investigate two dynamic workload balancing methods that focus on our database allocation strategy. Preliminary results show that we achieve both a balanced workload and very good performances.

The remainder of this paper is organized as follows: we discuss BLAST parallelization, particularly workload balancing, in Section 2. Then, in Section 3, we present our main ideas about database allocation and a BLAST parallelization approach. Some preliminary results are given in Section 4. Finally, Section 5 lists contributions, conclusions and possible extensions.

2 Motivation

The work in [4] presents a detailed discussion and implementation results regarding database distribution design in order to execute BLAST in workstation clusters. We have compared methods that run on both partitioned and replicated database situations. Both input sequences (query) allocation and database distribution strategies are shown to be important in order to improve the parallel execution of BLAST processes.

We have also evaluated a few different workload balancing methods that may be used when running BLAST in parallel. These include strategies that consider the total number of sequences allocated to each node, methods that estimate the total execution time at each site and also on-demand approaches, that distribute tasks (queries and data) to sites whenever these become idle [4].

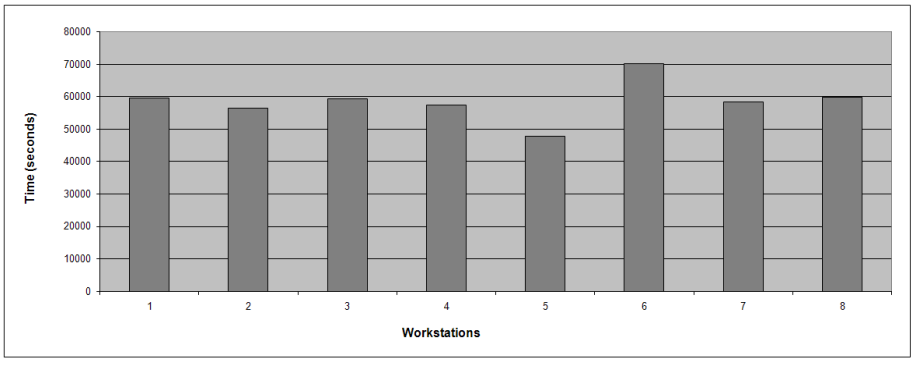


Fig. 1. Uneven Workload for Equal DB sizes

Due to the sensibility of each strategy with respect to multiple parameters, different techniques may be applied in order to deal with data skew factors. Particularly, dynamic issues like *similarity skew*, which cannot be detected before the actual execution [5], must be taken into account.

We show in Figure 1 an example of BLAST parallel execution on a 8-nodes homogeneous cluster, with a *pure* fragmented database configuration - distinct fragments at approximately the same size at each node. Some of the input sequences were randomly taken from the same database. One could expect a "perfect" load balancing when considering only database sizes. However, query decomposition assigns distinct input sequences to each node. As some of the sequences are more similar to the database sequences than others, the alignment process takes longer to finish. This similarity (or alignment) skew generates a clear uneven distribution of tasks.

There are many works that focus on BLAST parallelization. MPI-BLAST [8] is widely accepted as the standard parallel BLAST tool. It copies all query sequences to each workstation, while the database is segmented for a demand driven delivery. For each idle workstation, a new database fragment is sent. MPI-BLAST developers argue that the process should use multiple fragments in order to balance the workload. However, it is not a trivial task to determine the optimal granularity for good performances. Moreover, concurrency is an important problem when workstations get fragments from the same master node.

Some other works discuss further issues. Just to mention a few, the authors in [1] propose a strategy called Dynamic BLAST. They show that one of the main problems for running BLAST in a distributed environment is related to assigning fragments with distinct sizes to each processing node. However, we have already shown that the similarity degree among is also a fundamental factor. The same question appears in [14], where the authors look forward to extend MPIBLAST for grid environments. The basic idea is that most biology research labs cannot afford to keep efficient PC clusters environments. Nevertheless, the MPIBLAST tool is not that straightforward to use: it needs frequent database re-formatting when either the database is updated, or the number of sites changes. Another solution is proposed in [12] and is not only dedicated to BLAST. There are fault-tolerant methods that could be considered as a load balancing strategy. However, the goal is to guarantee completeness, not necessarily with best performances.

The work in [10] modifies BLAST source code in order to minimize memory utilization and execute parallel disk access. There are machine clusters, each containing a database replica and a fragment of the query sequences. BLAST code is adapted to enable every machine in the cluster to access distinct fragments from a single database non-physically fragmented, and assigned to only one machine, called coordinator. All machines in a given cluster have the same input sequences. Therefore, it is a static load balancing technique that, among others, does not take into account similarity skew.

Many other proposals exist in the literature. Our work here brings a somewhat distinct point of view, focusing on a database-approach (database distribution design, I/O parallelism and query execution) to improve BLAST evaluation

in clusters or Grids. In the next section we present our strategy for BLAST parallelization, which considers dynamic load balancing to obtain good overall performances.

3 BLAST Parallelization Approach

In this paper we propose an approach to execute BLAST on distributed and autonomous environments, where the database is fully replicated at each site. Each copy is then physically fragmented at each site, where distinct sets of fragments, called primary copies, are mainly responsible for local BLAST processing. Figure 2 illustrates this our database allocation.

The main idea underlying this database assignment to processing nodes is twofold: on one hand, fragmentation enables a distributed execution and will be very effective when there is an even workload. When load unbalancing is detected, due to similarity skew or any other reason (e.g. a broken connection to one node), all other non-primary fragments already available at each site are then considered to achieve the complete execution. On the other hand, as the database is also replicated, it enables a correct BLAST execution regarding statistical issues and parametrization.

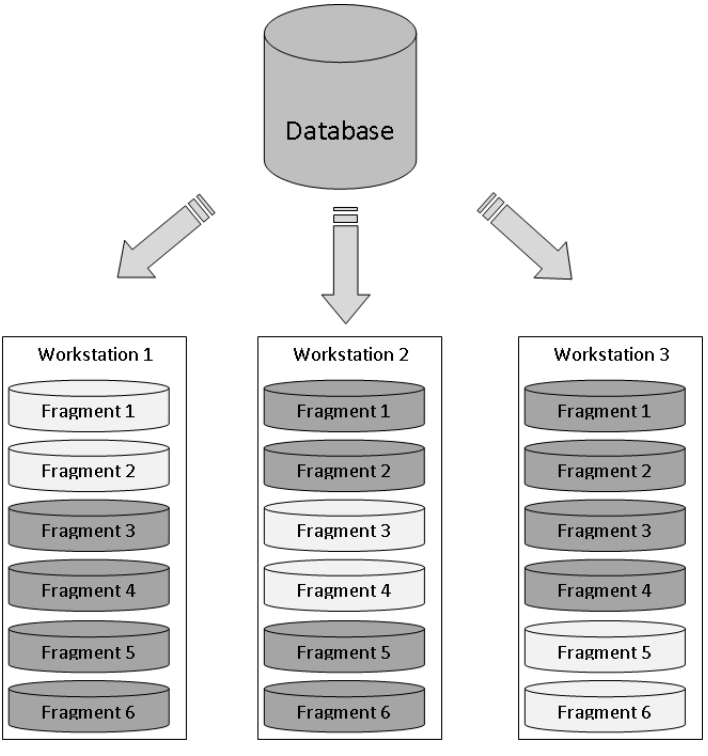


Fig. 2. Database replicated on 3 sites, each with their own primary fragments

The physical fragmentation method may vary. We have mostly used a fragmentation process that aims at obtaining fragments with approximately the same size. As we have already shown that equal database sizes are not sufficient for an even distribution of parallel tasks, we could have considered as well a simple round-robin database distribution [6].

At least two dynamic load balancing techniques may be applied for running BLAST in a cluster of workstations. A *demand-driven* approach, where processing sites ask for new tasks (input sequences and database fragments subset) when become idle and a *task stealing* strategy - where a site may execute part of a job originally assigned to another site. In this paper we argue that with these rather simple yet effective strategies, we make better use of the available resources in distributed environments. Both strategies are briefly explained in Figures 3 and 4.

3.1 Demand-Driven Approach

We will consider here that the database is replicated and contains primary fragments. The main goal of the demand-driven approach is to send new tasks as soon as given workstation becomes idle. This way we should make better use of the available resources.

A task is defined as a set of input (query) sequences and one (or more) fragments from the database. In order guarantee completeness of the execution, we may consider the following notation for a task sent, and assigned to a workstation:

Expression 1
$T(x : y)(z : w)$
$\forall x, y, z \text{ and } w; \;\; x \leq y \text{ and } z \leq w.$

Variables x and y represent the range of input sequences, and z and w the range within fragments. For example, the task T(1 : 5),(6 : 8) means: BLAST execution of sequences 1 to 5 checked with all fragments 6, 7 and 8. We define a range for a given task, rS for sequences and rF for fragments. In the previous example, the range of query sequences is $rS = 5$ and of fragments is $rF = 3$.

The initial task at each workstation has $x = 1$ and y is a parameter that defines the number of sequences per task. When the task is completed, the corresponding (partial) result is sent to the coordinating site and the processing node becomes idle, waiting for a new task. This process is repeated, with tasks being allocated to idle workstations until all tasks are finished.

We should observe that, with this strategy, a task is divided among many others, and these are executed in parallel. If we consider a complete BLAST processing as a single task, we can represent it as T(1 : n),(1 : m), where n is the number of input sequences and m the number of fragments. The subtasks can be defined as follows:

Let $\Delta n = \lfloor n/rS \rfloor, \Delta m = \lfloor m/rF \rfloor$

For the sake of simplicity, we consider that the remainder of both n/rS and m/rF is zero.

Expression 2

$T(1:n)(1:m)$

$\equiv$

$$\sum_{j=0}^{\Delta m-1} \sum_{i=0}^{\Delta n-1} T(i*rS+1:(i+1)*rS)(j*rF+1:(j+1)*rF)$$

This way one can verify if the whole execution has finished by checking if the complete task T(1 : n),(1 : m), or the equivalent expression, is executed. We illustrate this situation in Figure 3, that shows a demand-driven approach. Initially, each node gets a task including input query sequences and primary fragments. Input sequences are compared with the available fragments and, for each comparison, the result is sent to the master node. If there are still tasks available, the managing site sends it to the idle station.

We may suppose that all tasks (T(1 : n)(z : w)) initially defined for a machine are completely executed. Thus, this machine may work on other tasks with different values for z and w. When the managing station finds out that all reports for the same input sequence compared to all fragments are available, another concurrent process is generated, and will be responsible to gather all results in a single output file. This final assembly basically merges the *hits* for each file, optimizing disk access. As soon as all sequences are built up, we may concatenate the results.

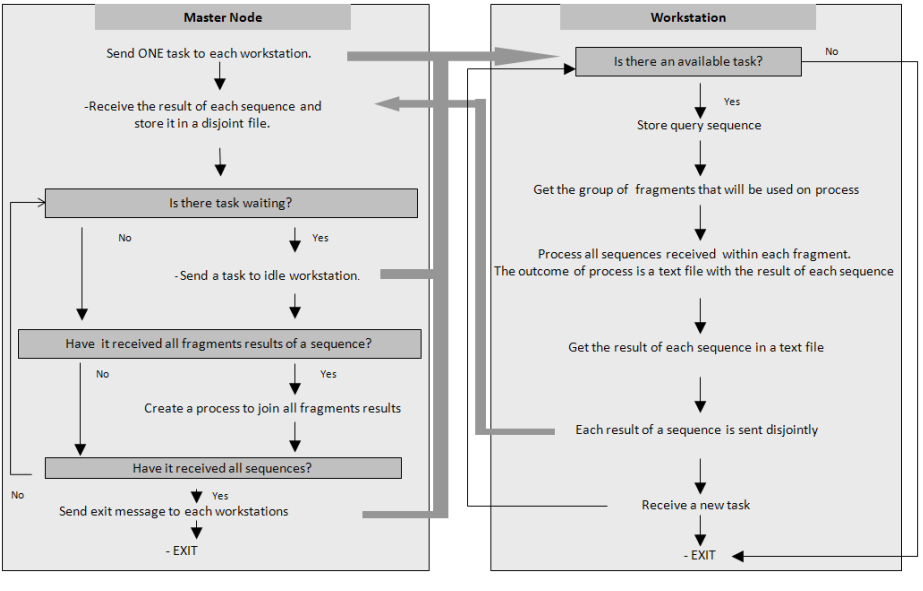


Fig. 3. Demand-driven load balancing strategy for BLAST

3.2 Task Stealing Strategy

In this approach, a task that represents the complete execution is divided by the number of available processing nodes, so that each node gets some part of the full task. When a machine finishes processing its initially assigned subtask, and there are machines still working, the idle machine may contribute getting (or "stealing") part of the subtask that is still active in another node. The goal here is to fine tune the initial unbalanced task allocation.

We can once more explain our strategy and its completeness through some expressions:

T(1 : n)(1 : m),

with n the number of sequences and m the number of fragments. The first step of the *task stealing* approach is to divide the task so that each subtask reaches one given workstation. We may represent these subtasks and their allocation as follows:

Expression 3

$T(1:n)(xi:xi+rF)$

where xi is the first fragment assigned to the processing site and rF the range of fragments considered. If we sum up all submitted tasks, we have a single task expressed by:

Let: $\Delta m = \lfloor m/rF \rfloor$

Once again, we consider zero the remainder of m/rF.

Expression 4

T(1 : n)(1 : m) $\equiv$

$$\sum_{i=0}^{\Delta m-1} T(1:n)(i*rF+1:(i+1)*rF)$$

Figure 4 illustrates the way the *task stealing* strategy works. Initially the coordinating station assigns one task to each machine. At each processing node, the input sequences are compared to a fragment, until all task sequences are over. At the end of the comparison, the results are sent to the managing station and some machines become idle. At this moment, the *Task Provider* module is called and it identifies the idle machine, the slowest among all active machines and the remainder of the original task to be executed that can be reassigned. Then, the slowest machine sends part of its execution to the idle station.

The coordinating station, as soon as it gets some partial result, verifies if the sequence was already compared to all fragments. In case of positive answer, another process is created to merge the results, similarly to the demand-driven strategy.

There might be 2 ways to reallocate a task: either the reallocation module decides that it will send only a few sequences. The second way would be to submit a full fragment that would be processed by all sequences at the idle machine.

We may observe that in this strategy, and due to the replicated database schema with primary fragments, there are no actual transmission of sequences

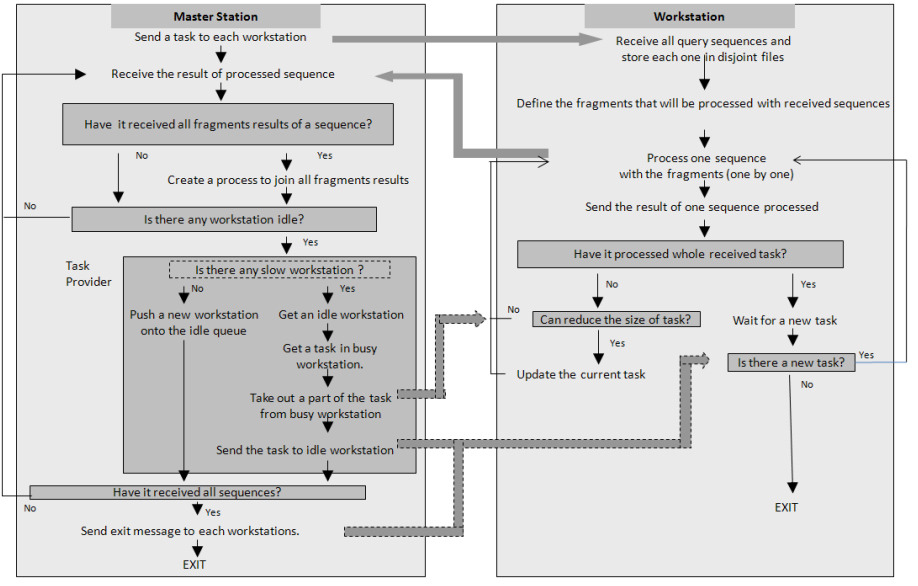


Fig. 4. Task stealing load balancing strategy for BLAST

or fragments during the re-assignment process. All data is already available at all machines. Only metadata messages are sent, informing and defining the task that will be executed.

4 Preliminary Experimental Results

In order to give an idea of the results obtained, we have observed (e.g. Figure 5) that both approaches (demand-driven and task-stealing) achieve much

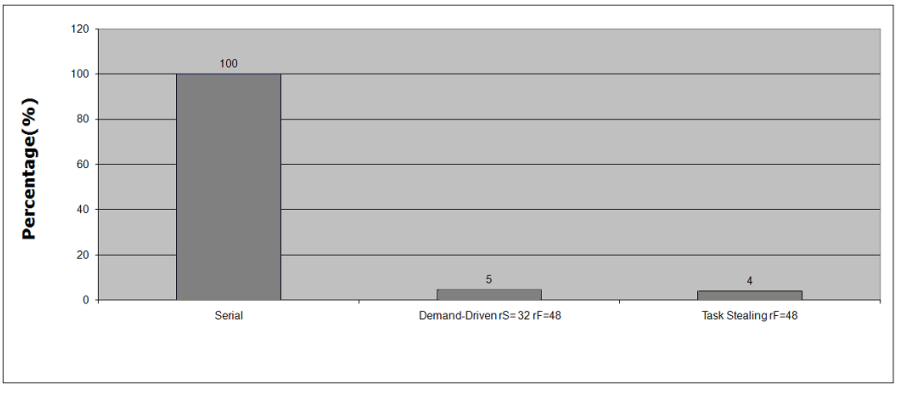


Fig. 5. Comparison among strategies

Fig. 6. Demand-driven Strategy and Load Balancing

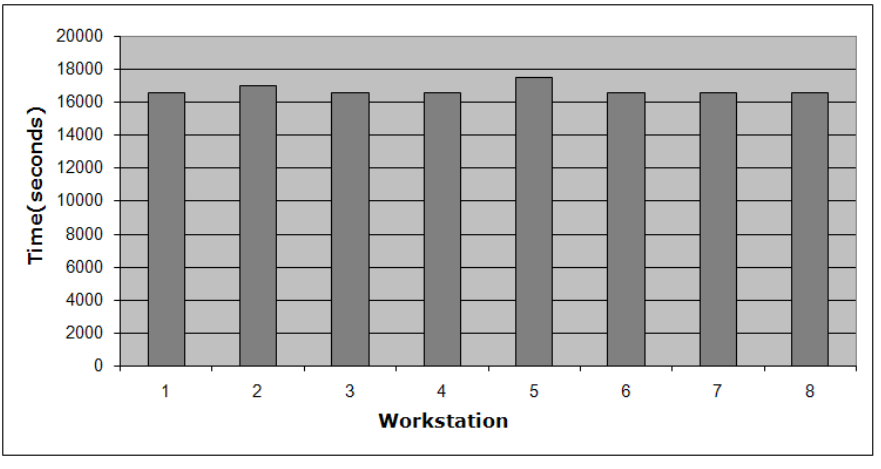


Fig. 7. Task-stealing Strategy and Load Balancing

better performances when compared to serial execution. We have compared our strategies to MPI-BLAST [8] and we have observed better performance as well.

We have executed MPI-BLAST with 2 fragment configurations, 48 and 96, and this helps the proposed load balancing method. Moreover, database fragments are all copied at each workstation, with no delays due to fragment copy from the coordinating site to workstations during executions. We still need to make an extensive study but our preliminary results comparing our strategies and MPI-BLAST have shown that our method is more efficient.

We should note that Figure 5 shows a comparison with MPI-BLAST (version 1.4) using 96 database fragments. Even though we have pushed all fragments into each workstation before actual execution, in order to reduce communication costs between the workstations and the master node, our strategies still outperform MPI-BLAST.

Our workload balancing strategies, *demand driven* and *task stealing*, initially split both the database and query sequences in order to process the entire

database. Each strategy has some parameters that the user can initially set. These parameters are very important, as they can make the strategy fit at different environments, either clusters or a grid, or even a more general P2P architecture. For these particular tests shown in Figure 5, the demand driven strategy sends runs 32 query sequences each time and the execution uses a fragmented database with 48 parts. Many other experimental results, mostly for cluster-based environments, can be found in [6].

Finally, besides efficiency, we show in Figure 6 and 7 how effective our approaches are regarding load balancing. Our performance results show that these dynamic strategies to balance the workload do not become an overhead for executing BLAST in parallel.

5 Conclusion

This paper discusses a database-oriented approach to deal with BLAST execution in distributed environments. Our goal is to fully exploit I/O parallelism and database partitioning, while guaranteeing correctness of BLAST execution. The main idea underlying replicated databases with primary fragments is that the workload may be initially distributed and, in case of any problems related to unavailability from single nodes to clusters, at least two load balancing methods could be considered.

In distributed, autonomous and independent environments like the grid, cooperation for parallel execution needs consistency (of program versions and data), otherwise the results would not be reliable. A replicated database is, then, more than natural, since genomic databases are usually kept at unique repositories in the web and users may download them at every new release. However, a parallel execution could be further exploited if at each site the BLAST program runs on only a fraction of the database - as we have done here.

An additional observation is related to the fact that we adopt a non-intrusive approach, that is, we make no changes to the specific (and usual) BLAST program used at each site. Indeed, we focus on a database point of view, investigating query processing and database allocation issues. For Grid environments, methods that modify particular source codes should be avoided. These and other data intensive challenges within a Grid are well discussed in [11].

References

1. Afgan, E., Sathyanarayana, P., Bangalore, P.: Dynamic Task Distribution in the Grid for BLAST. In: Procs. IEEE Intl. Conference on Granular Computing, pp. 554–557 (2006)
2. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: A Basic Local Alignment Search Tool. Journal of Molecular Biology 215, 403–410 (1990)
3. Chen, S.-N., Tsai, J.J.P., Huang, C.-W., Chen, R.-M., Lin, R.C.: Using Distributed Computing Platform to Solve High Computing and Huge Data Processing Problems in Bioinformatics. In: Procs. IEEE Intl. Symposium on Bioinformatics and Bioengineering (BIBE), pp. 142–148 (2004)

4. Costa, R.L.D.C., Lifschitz, S.: Database Allocation Strategies for Parallel BLAST Evaluation on Clusters. Distributed and Parallel Databases 13(1), 99–127 (2003)
5. Costa, R.L.D.C., Lifschitz, S.: Skew Handling for Parallel BLAST Processing. In: II Brazilian Workshop on Bioinformatics, pp. 173–176 (2003)
6. de Sousa, D.X.: Workload Balancing Strategies for BLAST Parallel Evaluation on Replicated Databases and Primary Fragments, MSc Dissertation, PUC-Rio Departamento de Informatica, p. 85 (2007), ftp://ftp.inf.pucrio.br/pub/docs/theses/07_MSc_sousa.zip
7. de Sousa, D.X., Lifschitz, S.: E-value Evaluation for BLAST Parallel Execution on Fragmented Databases, Tecnical Report MCC 17/07, PUC-Rio Departamento de Informatica, p.16 (2007), ftp://ftp.inf.pucrio.br/pub/docs/techreports/07_17_sousa.pdf
8. mpiBLAST, http://www.mpiblast.org/
9. NCBI-BLAST, http://www.ncbi.nlm.nih.gov/BLAST
10. Oehmen, C., Nieplocha, J.: ScalaBLAST: A Scalable Implementation of BLAST for High-Performance Data-Intensive Bioinformatics Analysis. IEEE Transactions of Parallel and Distributed Systems 17, 740–749 (2006)
11. Pacitti, E., Valduriez, P., Mattoso, M.: Grid Data Management: Open Problems and New Issues. Journal of Grid Computing 5, 273–281 (2007)
12. Sun, Y., Zhao, S., Yu, H., Gao, G., Luo, J.: ABCGrid: Application for Bioinformatics Computing Grid. Bioinformatics (Applications Note) 23(9), 1175–1177 (2007)
13. WU-BLAST, http://blast.wustl/edu/
14. Yang, C.-T., Han, T.-F., Kan, H.-C.: G-BLAST: a Grid-Based Solution for mpiBLAST on Computational Grids. In: Procs. IEEE TENCON 2007, pp. 1–5 (2007)

Testing Architectures for Large Scale Systems*

Eduardo Cunha de Almeida**, Gerson Sunyé, and Patrick Valduriez

INRIA - University of Nantes
eduardo.almeida@univ-nantes.fr

Abstract. Typical distributed testing architectures decompose test cases in actions and dispatch them to different nodes. They use a central test controller to synchronize the action execution sequence. This architecture is not fully adapted to large scale distributed systems, since the central controller does not scale up. This paper presents two approaches to synchronize the action execution sequence in a distributed manner. The first approach organizes the testers in a B-tree manner synchronizing through messages exchanged among parents and children. The second approach synchronizes through gossiping messages exchanged among consecutive testers. We compare these two approaches and discuss their advantages and drawbacks.

1 Introduction

Current Grid solutions focus on data sharing and collaboration for statically defined virtual organizations with powerful servers. They cannot be easily extended to satisfy the needs of dynamic virtual organizations such as professional communities where members contribute their own data sources, perhaps small ones but in high numbers, and may join and leave the Grid at will. In particular, current solutions require heavy organization, administration and tuning which are not appropriate for large numbers of small devices.

Peer-to-Peer (P2P) techniques which focus on scalability, dynamism, autonomy and decentralized control can be very useful to Grid data management. The synergy between P2P computing and Grid computing has been advocated to help resolve their respective deficiencies [13]. For instance, Narada [14], P-Grid [2] and Organic Grid [5] develop self-organizing and scalable Grid services using P2P interactions. The Grid4All European project [8] which aims at democratizing the Grid is also using P2P techniques. As further evidence of this trend, the Global Grid Forum has recently created the OGSA-P2P group [4] to extend OGSA for the development of P2P applications.

Grid and P2P systems are becoming key technologies for software development, but still lack an integrated solution to validate the final software, in terms

* Work partially funded by CAPES-COFECUB (DAAD project), CNPq-INRIA (GriData project), French ANR Massive Data (Respire project) and the European Strep Grid4All project.

** Supported by the Programme Alβan, the European Union Programme of High Level Scholarships for Latin America, scholarship no. E05D057478BR.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 555–566, 2008.

of correctness and security. Although Grid and P2P systems usually have a simple public interface, the interaction between nodes is rather complex and difficult to test. For instance, distributed hash tables (DHTs) [23,25], provide only three public operations (insert, retrieve and lookup), but need very complex interactions to ensure the persistence of data while nodes leave or join the system. Testing these three operations is rather simple. However, testing that a node correctly transfers its data to another node before leaving requires the system to be in a particular state. Setting a system into a given state requires the execution of a sequence of actions, corresponding to the public operation calls as well as the requests to join or leave the system, in a precise order. The same rationale can be applied to data grid management systems (DGMS) [16].

In Grid and P2P systems, actions can be executed in parallel, on different nodes. Thus, an action can run faster or slower depending on the node computing power. Synchronization is then needed to ensure that a sequence of actions of a test case is correctly executed. For instance, suppose a simple test case where a node removes a value previously inserted by another node. To correctly execute this test case, the execution must ensure that the insertion is performed before the removal. Typical testing architectures [11,19,28,10] use a central test controller to synchronize the execution of test cases on distributed nodes. This approach is not fully adapted for large scale systems, since it does not scale up while testing on a large number of nodes.

In this paper, we propose two different architectures to synchronize the execution of test cases in distributed systems. The first architecture organizes the testers in a balanced tree [3] (B-Tree) structure where the synchronization is performed from the root to the leaves. The second approach uses gossiping messages among testers, reducing communications among the testers responsible to execute consecutive test case actions. Since both architectures do not rely on a central coordinator they scale up correctly.

This paper is organized as follows. Section 2 discusses the related work. In Section 3, we introduce some fundamental concepts in software testing. In Section 4, we discuss the centralized approach in detail, and present two distributed approaches and their trade-off. In Section 5, we present some initial results through implementation and experimentation. Section 6 concludes.

2 Related Work

In the context of distributed systems testing, different approaches can be used either to schedule or control the execution of test case actions. However, these approaches neither ensure the correct execution of a sequence of actions, nor scale up with the system under test.

Typical Grid task scheduling techniques, using centralized [9,22] or distributed [26] approaches are not suitable for system testing, since they do not follow the same objectives. While the objective of task scheduling is to dispatch tasks to the most available nodes, the objective of the test controller is to dispatch tasks (i.e. test case actions) to predefined nodes. Moreover, task scheduling focus on

parallel execution, while the test controller must ensure the execution sequence of tasks.

In the domain of distributed system testing, Kapfhammer [19] describes an approach that distributes the execution of test cases. The approach is composed of three components. The first component is the *TestController* which is responsible to prepare the test cases and to write them into the second component called *TestSpace*, that is a storage area. The third component, called *TestExecutor*, is responsible to consume the test cases from the *TestSpace*, to execute them, and to write the results back into the *TestSpace*. A solution based on this approach, called GridUnit, is presented by Duarte et al. [11,12]. The main goal of GridUnit is to deploy and to control unit tests over a grid with minimum user intervention aiming to distribute the execution of tests to speed up the testing process. To distribute the execution, different test cases can be executed by different nodes. However, a single test case is executed only by a single node. Unlike our approach, in GridUnit, it is not possible to write more complex test cases where different nodes execute different actions of the same test case. Moreover, GridUnit does not handle node failure, and this may assign a false-negative verdict to test cases.

Ulrich et al. [28] describe two test architectures for testing distributed systems using a global tester and a distributed tester. The distributed tester architecture, which is close to our algorithm, divides test cases in small parts called *partial test cases* (PTC). Each PTC is assigned to a distributed tester and can be executed in parallel to another PTC with respect to a function that controls the mutual exclusivity. The behavior of the distributed testers is controlled by a *Test Coordination Procedure* (TCP) which coordinates the PTCs execution by synchronization events. Through this approach different nodes can execute different actions, however, the same action can not be executed in parallel by different nodes. Such kind of execution can be very useful in certain kinds of tests like performance or stress testing, where several nodes insert data at the same time.

3 Testing Large Scale Distributed Systems

Software testing aims at detecting faults and usually consists of executing a system with a suite of *test cases* and comparing the actual behavior (e.g. the observable outputs) with the expected one. The objective of a test case is thus both to exercise the system and to check whether an erroneous behavior occurs. The first aspect relates to test inputs (or test scenario) generation, which may be guided by various test criteria (control/based-flow based coverage criteria, specification-based coverage criteria). The second aspect concerns the way the verdict is obtained (often call the 'oracle'), which means a mechanism to check whether the execution is correct (e.g. embedded assertions).

The role of the oracle is to compare the output values with the expected ones and to assign a verdict to the test case. If the values are the same, the verdict is pass. Otherwise, the verdict is fail. The verdict may also be inconclusive, meaning that the test case output is not precise enough to satisfy the test intent and the

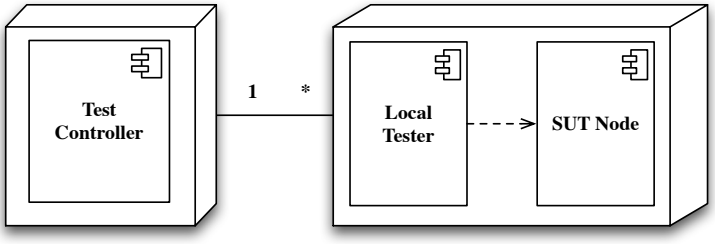


Fig. 1. Typical Centralized Tester Architecture

test must be done again. There are different sorts of oracles: assertions [27], value comparison, log file analysis, manual, etc.

A testing technique thus includes test criteria, test cases generation techniques and mechanisms for obtaining the oracle. In this paper, we roughly define a test case as being composed of a name, an intent, a sequence of input data and the expected outputs.

Grid and P2P systems are distributed applications, and should be firstly tested using appropriate tools dedicated to distributed system testing. Distributed systems are commonly tested using conformance testing [24]. The purpose of conformance testing is to determine to what extent the implementation of a system conforms to its specification. The tester specifies the system using Finite State Machines [7,15,6], Labeled Transition Systems [17,21,18] and uses this specification to generate a test suite that is able to verify (totally or partially) whether each specified transition is correctly implemented. The tester then observes the events sent among the different nodes of the system and verifies that the sequence of events corresponds to the state machine (or to the transition system).

The classical architecture for testing a distributed system, illustrated by the UML deployment diagram presented in Figure 3, consists of a *test controller* which sends the test inputs, controls the synchronization of the distributed system and receives the outputs (or local verdicts) of each node of the system under test (SUT). In many cases, the distributed system under test is perceived as a single application and it is tested using its external functionalities, without considering its components (i.e. black-box testing). The tester in that case must interpret results which include non-determinism since several input/outputs orderings can be considered as correct.

The observation of the outputs for a distributed system can also be achieved using the traces (i.e. logs) produced by each node. The integration of the traces of all nodes is used to generate an event timeline for the entire system. Most of these techniques do not deal with large scale systems, in the sense they target a small number of communicating nodes. In the case of Grid and P2P systems, the tester must observe the remote interface of peers to observe their behavior and she must deal with a potentially large number of peers. Writing test cases is then particularly difficult, because non-trivial test cases must execute actions on different peers. Consequently, synchronization among actions is necessary to control the execution sequence of the whole test case.

Analyzing the specific features of Grid and P2P system, we remark that they are distributed systems, but the existing testing techniques for distributed systems do not address the issue of synchronization when a large number of nodes are involved. Moreover, the typical centralized tester architecture can be a bottleneck when building a testing framework for these systems.

3.1 Test Case Sample

A test case noted τ is a tuple $\tau = (A^\tau, T^\tau, V^\tau, S^\tau)$ where $A^\tau \subseteq A$ is an ordered set of m actions $\{a_0, \ldots, a_m\}$, T^τ a set of n testers $\{t_0, \ldots, t_n\}$, V^τ is a set of local verdicts and S^τ is a schedule.

The schedule is a map between actions and sets of testers, where each action corresponds to the set of testers that execute it.

A test case action is a tuple $a_i^\tau = (\Psi^a, \theta^a, T_i^a)$ where Ψ^a is a set of instructions, θ^a is the interval of time in which a should be executed and $T_i^a \subseteq T$ is a subset of testers $\{t_0^\tau, \ldots, t_n^\tau\}$ that execute the action. The are three different kinds of instructions: (i) calls to the peer application public interface; (ii) calls to the tester interface and (iii) any statement in the test case programming language. The time interval θ ensures that actions do not wait eternally for a blocked peer.

Let us illustrate these definitions with a simple distributed test case (see example 1). The aim of this test case is to detect errors on a Distributed Hash Table (DHT) implementation. More precisely, it verifies if a node successfully resolves a given query, and continues to do so in the future.

Example 1 (Simple test case).

Action	**Nodes**	**Instructions**
(a_1)	0,1,2	Join the system;
(a_2)	2	Insert the string "One" at key 1; Insert the string "Two" at key 2;
(a_3)	*	Pause;
(a_4)	0	Retrieve data at key 1; Retrieve data at key 2;
(a_5)	1	Leave the system;
(a_6)	0	Retrieve data at key 1; Retrieve data at key 2;
(a_7)	0,2	Leave the system;
(v_0)	0	Calculate a verdict;

This test case involves three testers $T^\tau = \{t_0, t_1, t_2\}$ managing seven actions $A^\tau = \{a_1, ..., a_7\}$ on three nodes $P = \{p_0, p_1, p_2\}$. The goal of the first three actions is to populate the DHT. The only local verdict is given by t_0. If the data retrieved by p_0 is the same as the one inserted by p_2, then the verdict is *pass*. If the data is not the same, the verdict is $fail$. If p_0 is not able to retrieve any data, then the verdict is *inconclusive*.

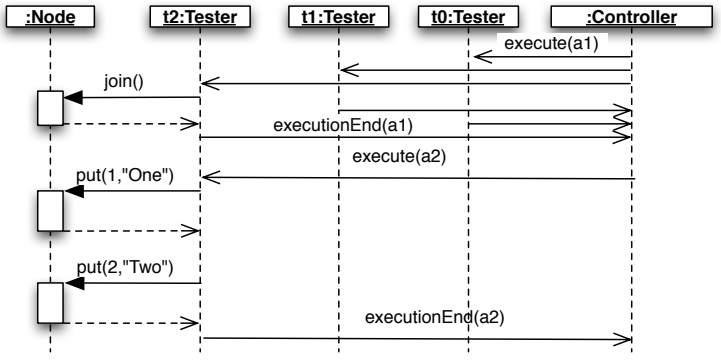


Fig. 2. Test case execution

The UML sequence diagram presented in Figure 2 illustrates the execution of the first two actions of the test case. First, the test controller asks all testers to execute action a_1. Then, each tester executes a set of instructions, interacting with the SUT. Before asking tester t_2 to execute action a_2, the test controller waits for the execution of a_1 to end. Once the execution of the test case is finished, all testers send their local verdicts to the test controller. The later compiles all local verdicts and assigns a verdict to the test case.

3.2 Problem Statement

In a centralized testing architecture, the test controller dispatches actions to a variable number of testers and waits for execution results from them. The controller must then maintain a bidirectional communication channel with all testers, excluding the use of a multicasting protocol, which is fast and scalable, but unidirectional. Multicasting could be used to dispatch efficiently actions to all testers, but not to receive the execution results.

The complexity of the execution algorithm is $O(n)$, meaning that the typical architecture for testing distributed systems, using a unique test controller, is thus not adapted for testing large scale distributed systems.

4 Architecture

In this section, we present two alternatives to the centralized test controller architecture.

4.1 B-Tree

The first architecture presented here consists of organizing testers in a B-Tree structure, similarly to the overlay network used by GFS-Btree [20]. The idea is to drop the test controller and use the tester that is the root of the tree to control the execution of test cases and to assign their verdict. When executing a

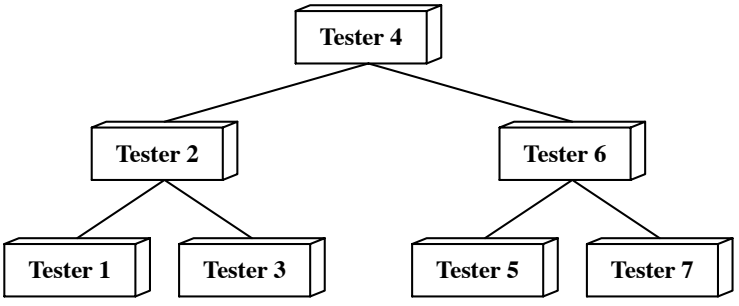


Fig. 3. B-Tree Architecture

test case, the root dispatches actions to its child testers, who dispatch actions to their children. Once an action is executed, the leaves send their results to their parents, until the root receives all results and can dispatch the next actions.

Figure 3 presents an example of tester organization using a B-Tree of order 1, where tester 4 is the root. Tester 4 will only communicate with testers 2 and 6. The leaves, 1, 3, 5 and 7 do not dispatch any action, they only send their results to their parents.

The order of the B-Tree is not fixed, it may vary according to the number of testers, which is known at the beginning of the execution. The goal is to have a well-proportioned tree, where the depth is equivalent to its order.

4.2 Gossiping

Besides the B-tree approach, the Gossiping is another solution to synchronize the execution of actions in a distributed manner. In Gossiping, we use the same architecture used by the B-Tree approach with a tester per node, however, the synchronization of actions is executed by gossiping the coordination messages among the testers.

The Gossiping approach has the following steps. First, any node p in the system P is designated to execute the first tester t_0. This tester will act as an identifier to all the other testers t_n that join the system. The identification follows an incremental sequence from 0 up to n and is used to select the actions a node should execute. Second, t_0 creates a multicast address for each test case action. Third, the decomposed test case is deployed through P and stored at each tester. Then, each tester verifies which actions it should execute and subscribes to the suitable multicast addresses. Finally, the testers responsible for the first action start the execution.

A tester can play two different roles during the test case execution:

- *Busy tester*. This tester executes an action a_i and gossips its completion to the multicast address of the next action a_{i+1}. Once it has sent a gossip, it becomes an *Idle tester*.
- *Idle tester*. This tester remains idle waiting the gossips from all the *Busy testers*. Once it receives all their gossips, then it becomes a *Busy tester*.

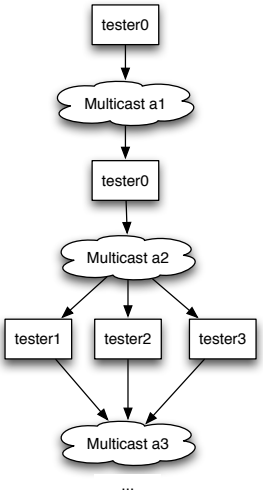


Fig. 4. Gossiping Architecture

The gossiping between these two types of testers guarantees the execution sequence of the whole test case.

We use the example 1 to illustrate this approach. Initially any node is chosen to be tester t_0. Then, the other nodes contact t_0 to receive an identifier n and subscribe to the suitable multicast addresses. For instance, if a tester receives $n = 1$, it subscribes to the addresses of a_1, a_3 and a_5. Figure 4 presents the first action a_1 being executed by testers $\{t_0, t_1, t_2\}$. Once the execution of a_1 is finished, the testers gossip the completion to the multicast address of the next action a_2. Once tester t_2 receives all three multicast messages, it executes a_2 gossiping in the end as well. This happens consecutively up to the last action a_7. Finally, each tester calculates a local verdict and sends it to t_0, which assigns a verdict of the entire test case.

5 Experimentation

When implementing PeerUnit [10], we have chosen a centralized architecture. This choice was due to its simplicity. However, as the performance evaluation shows, this architecture may limit the number of testers and thus, the number of nodes of the system under test. We intend to implement the two architectures presented here and evaluate their performance using the same experimentation.

For our experiments, we implemented the test controller in Java (version 1.5), and we use two clusters of 64 machines[1] running Linux. In the first cluster, each machine has 2 Intel Xeon 2.33GHz dual-core processors. In the second cluster, each machine has 2 AMD Opteron 248 2.2GHz processors. Since we can have full control over these clusters during experimentation, our experiments are

[1] The clusters are part of the Grid5000 project [1].

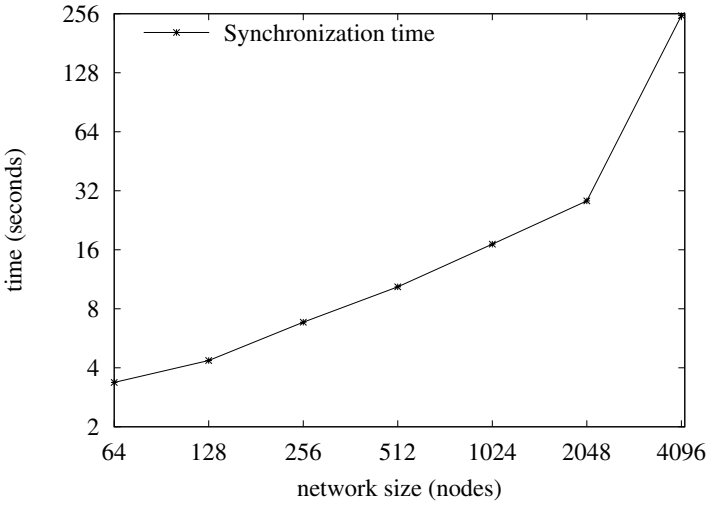


Fig. 5. Synchronization algorithm evaluation

reproducible. The implementation produced for this paper can be found in our web page[2]. We allocate the peers equally through the nodes in the clusters up to 8 peers per machine. In all experiments reported in this paper, each node is configured to run in a single Java VM.

5.1 Centralized Test Controller

In order to measure the response time of action synchronization, we submitted a fake test case, composed of empty actions through a different range of testers. Then, for each action, we measured the whole execution time, which comprises remote invocations, execution of empty actions and confirmations.

The evaluation works as follows. We deploy the fake test case through several testers. The testers register their actions with the coordinator. Once the registration is finished, the coordinator executes all the test case actions inside and measures their execution time. The evaluation finishes when the execution of all actions is over.

The fake test case contains 8 empty actions (we choose this number arbitrarily) and is executed until a limit of 2048 testers running in parallel. Figure 5 presents the response time for action synchronization for a varying number of testers. The response time grows linearly with the number of nodes as expected for an algorithmic complexity of $O(n)$.

5.2 Discussion

The centralized test controller showed a linear performance in terms of response time. Although this result was expected, its implementation is easy and can

[2] Peerunit project, http://peerunit.gforge.inria.fr

be even used while testing in small scale environments. Our target, however, is testing in large scale environments.

The B-tree approach relies on communications between parents and children in order to reduce the communication cost and avoid the use of a centralized test controller either. In one hand, this approach scales up better than any approach described in this paper. In the other hand, it has two problems. First, a tree structure has to be built in the beginning of each execution. Second, a new action will start the execution in the root earlier than in the leaves.

The Gossiping approach can also be easy to implement and has two main advantages. First, it does not require any particular node structure (e.g. B-Tree or ring). Second, the communication can be implemented using multicast messages in order to reduce the communication cost. A weakness of this approach is the execution of consecutive actions by all the testers which requires $O(n)$ gossiping messages.

As a comparison, using the example 1 and considering that the worst case happens between actions a_2 and a_3, the B-Tree approach would need two messages to coordinate the test while the Gossiping would need three messages. In one hand, the B-Tree uses round-trip messages while the gossiping uses multicast. In the other hand, in the B-Tree a tester waits a maximum of two messages from its children while in gossiping the same tester would wait n multicast messages, $n = 3$ in this case.

6 Conclusion

In this paper, we presented two synchronization approaches to control the execution of test cases in a distributed manner using P2P techniques. Since both approaches do not rely on a central coordinator they scale up correctly.

We discuss the approaches, including the centralized, presenting their strengths and weaknesses.

We currently implement a testing tool that supports both distributed approaches for later evaluation.

References

1. Grid5000 project, `http://www.grid5000.fr/`
2. Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R.: P-grid: a self-organizing structured p2p system. SIGMOD Rec. 32(3), 29–33 (2003)
3. Bayer, R., McCreight, E.M.: Organization and maintenance of large ordered indices. Acta Inf. 1, 173–189 (1972)
4. Bhatia, K.: Peer-to-peer requirements on the open grid services architecture framework. OGF Informational Documents (INFO) GFD-I.049, OGSA-P2P Research Group (2005)
5. Chakravarti, A.J., Baumgartner, G., Lauria, M.: The organic grid: self-organizing computation on a peer-to-peer network. IEEE Transactions on Systems, Man, and Cybernetics, Part A 35(3), 373–384 (2005)

6. Kai Chen, Fan Jiang, and Chuan dong Huang. A new method of generating synchronizable test sequences that detect output-shifting faults based on multiple uio sequences. In *SAC*, pages 1791–1797, 2006.
7. Chen, W.-H., Ural, H.: Synchronizable test sequences based on multiple uio sequences. IEEE/ACM Trans. Netw. 3(2), 152–157 (1995)
8. Grid4All Consortium. Grid4all: democratize the grid. World Wide Web electronic publication (2008)
9. da Silva, D.P., Cirne, W., Brasileiro, F.: Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 169–180. Springer, Heidelberg (2003)
10. de Almeida, E.C., Sunyé, G., Valduriez, P.: Action synchronization in p2p system testing. In: EDBT Workshops (2008)
11. Duarte, A., Cirne, W., Brasileiro, F., Machado, P.: Using the computational grid to speed up software testing. In: Proceedings of the 19th Brazilian Symposium on Software Engineer (2005)
12. Duarte, A., Cirne, W., Brasileiro, F., Machado, P.: Gridunit: software testing on the grid. In: ICSE 2006: Proceeding of the 28th international conference on Software engineering, pp. 779–782. ACM Press, New York (2006)
13. Foster, I.T., Iamnitchi, A.: On death, taxes, and the convergence of peer-to-peer and grid computing. In: Frans Kaashoek, M., Stoica, I. (eds.) IPTPS 2003. LNCS, vol. 2735, pp. 118–128. Springer, Heidelberg (2003)
14. Hancock, J.: The NaradaBrokering project. World Wide Web electronic publication (2008)
15. Hierons, R.M.: Testing a distributed system: generating minimal synchronised test sequences that detect output-shifting faults. Information and Software Technology 43(9), 551–560 (2001)
16. Jagatheesan, A., Rajasekar, A.: Data grid management systems. In: SIGMOD 2003: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, p. 683. ACM, New York (2003)
17. Jard, C.: Principles of distribute test synthesis based on true-concurrency models. Technical report, IRISA/CNRS (2001)
18. Jard, C., Jéron, T.: Tgv: theory, principles and algorithms: A tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems. Int. J. Softw. Tools Technol. Transf. (2005)
19. Kapfhammer, G.M.: Automatically and transparently distributing the execution of regression test suites. In: Proceedings of the 18th International Conference on Testing Computer Software, Washington, D.C. (June 2001)
20. Li, Q., Wang, J., Sun, J.-G.: Gfs-btree: A scalable peer-to-peer overlay network for lookup service. In: Li, M., Sun, X.-H., Deng, Q.-n., Ni, J. (eds.) GCC 2003. LNCS, vol. 3032, pp. 340–347. Springer, Heidelberg (2004)
21. Pickin, S., Jard, C., Traon, Y.L., Jéron, T., Jézéquel, J.-M., Le Guennec, A.: System test synthesis from UML models of distributed software. In: ACM - 22nd IFIP WG 6.1 International Conference Houston on Formal Techniques for Networked and Distributed Systems (2002)
22. Qin, X., Jiang, H.: Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems. In: ICPP, pp. 113–122 (2001)
23. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenkern, S.: A scalable content-addressable network. In: ACM SIGCOMM (2001)
24. Schieferdecker, I., Li, M., Hoffmann, A.: Conformance testing of tina service components - the ttcn/ corba gateway. In: IS&N, pp. 393–408 (1998)

25. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peertopeer lookup service for internet applications. ACM, New York (2001)
26. Subramani, V., Kettimuthu, R., Srinivasan, S., Sadayappan, P.: Distributed job scheduling on computational grids using multiple simultaneous requests. In: 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002), Edinburgh, Scotland, UK, July 23-26 (2002)
27. Traon, Y.L., Baudry, B., Jézéquel, J.-M.: Design by contract to improve software vigilance. IEEE Trans. Software Eng. 32(8), 571–586 (2006)
28. Ulrich, A., Zimmerer, P., Chrobok-Diening, G.: Test architectures for testing distributed systems. In: Proceedings of the 12th International Software Quality Week (1999)

Data Replication and the Storage Capacity of Data Grids

Silvia Figueira[1] and Tan Trieu[2]

[1] Department of Computer Engineering
Santa Clara University
Santa Clara, CA 95053-0566, USA
sfigueira@scu.edu
[2] SGI, 1140 E. Arques Avenue
Sunnyvale, CA 94085, USA
tantrieu@sgi.com

Abstract. Storage is undoubtedly one of the main resources in data grids, and planning the capacity of storage nodes is an important step in any data-grid design. This paper focuses on storage-capacity planning for data grids. We have developed a tool to calculate, for a specific scenario, the minimum capacity required for each storage node in a grid, and we have used this tool to show that different strategies used for data replication may lead to different storage requirements, affecting the storage-capacity planning.

Keywords: Data Grids, Storage, Capacity Planning, Data Replication.

1 Introduction

Grid computing addresses the challenge of coordinating resource sharing and problem solving in dynamic, multi-institutional virtual organizations [3]. Though we often think of processor power as a primary motivation for grid computing, access to distributed data is typically as important as access to distributed computational resources. In fields such as high-energy physics, biology and medical image processing, and earth observations, experiments can produce massive amounts of data, on the scale of petabytes per year [4]. The global sharing of such large amounts of data introduces access, processing, and distribution difficulties. With the massive size of the data, the task of managing the data quickly becomes a problem. Data grids have been developed to facilitate the efficient storage and quick distribution of this data [11].

A data grid connects a collection of geographically distributed computer and storage resources, enabling users to share data and other resources in a seamless fashion. The European Data Grid project is an effort to achieve the vision of uniform and transparent access to data and computing resources [10]. The vision is a computing environment in which a scientist who wants to run a computationally intensive process on a huge data set has several options. If there are sufficient computing resources on the local network, the scientist should be able to download the data to a local destination and perform the job locally. If the local site lacks the necessary computing resources, the scientist could choose to off-load the processing to the data site.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 567–575, 2008.

However, if the data site is under heavy load, the scientist could also request that the data be replicated to another site and run the job on that site.

The design of a data grid requires forethought concerning the capacity of resources in order to avoid over-provisioning and/or under-provisioning at different locations of the grid. Capacity planning is defined as the process of assessing the cost/benefit of a system configuration before actually building it [1, 5, 6]. A useful prediction necessarily considers the evolution of the workload on the data grid. With capacity planning, system administrators can input different network topologies, while varying the number of resource requests and other parameters (such as network link capacities, node storage capacities, and data replication strategies) in order to fine-tune the data grid's resource utilization.

Since one of the main resources in data grids is storage, planning the capacity of storage nodes is a key part of a data-grid design. This paper focuses on storage-capacity planning for data grids. We have developed a tool to calculate, for a specific scenario, the minimum capacity required for each storage node in a grid. The scenarios are defined by the network topology and the workload characteristics. The tool combines the topology with the load characteristics to produce guidelines on the minimum storage capacity required in each storage node to provide enough space to the workload considered. We have used this tool to compare data-replication strategies and show that they affect the storage required.

This paper is organized as follows. Section 2 discusses data replication. Section 3 presents a strategy for calculating storage capacity, taking into account data replication. Section 4 shows how data replication affects the need for storage in data grids. Section 5 concludes.

2 Data Replication

Data replication is a technique used in grid computing to both decrease access time to data and increase fault tolerance. Replication is especially important when considering requests for transferring massive amounts of data between geographically distant locations, which consume large amounts of bandwidth and are delayed by possibly high latencies [8, 9].

With the goal of reducing access latency and bandwidth consumption, recent research has addressed the usefulness of creating replicas to distribute data among scientists within a grid environment. Data replication can be managed statically or dynamically. Though static replication plans do improve load balancing and reliability, it does not adapt to changes in load behavior. Since data grids are intended for a global computing environment, where variable data-access patterns are expected, dynamic replication is preferred. With dynamic replication, the replication strategies adapt to changes in user behavior, making heavy use of locality in determining where files should be replicated. Decisions are made based on the notion that recently accessed files are more likely to be accessed again (temporal locality), files recently accessed by a node are likely to be accessed by nearby nodes (geographical locality), and files near recently accessed files are likely to be accessed (spatial locality).

In [8], the authors discussed and evaluated six replication/caching strategies:

- No replication or caching.
- Best client: Nodes maintain an access history for each file, and when a threshold number of accesses is reached, review the history to determine the replica destination.
- Cascading replication: Take advantage of the hierarchical layout of a grid by replicating the requested file along the path from server to requester. A replica is made to the next node on the path to the requester only after a threshold number of requests have been recorded at the data site.
- Plain caching: Simply store a local copy on the requesting node.
- Caching plus cascading replication: The requester caches files locally and the server periodically identifies popular files to propagate down the network hierarchy.
- Fast spread: Requested files are stored at each node along the path from server to client.

The results of the study indicate that among the six replication/caching strategies, cascading and fast spread provide the best overall performance, where performance is measured as savings in latency and bandwidth consumption. The distinction between the two strategies is that fast spread works well in a network where users exhibit total randomness in accessing data. Cascading is the best option when access patterns exhibit geographical locality.

3 Calculating Storage Capacity

The data-grid capacity planner will take two main inputs: (1) the network topology, which specifies how the storage nodes are connected, and (2) a synthetic trace, which simulates requests for data. Based on these two inputs, it simulates the requests on the topology, tracking the storage usage at each node, according to a replication strategy. The main goal of the tool is to calculate the maximum storage capacity needed at each storage node after all the requests were processed. The output is a table detailing storage consumption at each node.

The requests will determine the data to be copied from the source to one or more nodes, according to the replication strategy. We propose the following scheme to characterize the requests:

- The source nodes, in which data is generated and from which it is disseminated, are determined initially. This information is provided by the user, who also determines the size of the data initially placed in each node and a number of tags to identify portions of each data. Each tag is associated with a size as well.
- Another parameter provided by the user is the lifetime of each request, which represents the time interval during which data should be useful for the destination node. This information could be a constant value or generated uniformly within a range provided by the user.
- Each request will have a source node, a destination node, a tag, and a lifetime.

The algorithm for calculating the capacity is shown below. After the initial data is assigned to each source node, as determined by the user, and each source node has its current capacity updated, the algorithm starts processing the requests. A request will identify a source, a tag, a destination, and a lifetime. According to the replication strategy, the algorithm will decide which nodes need to receive the data. Note that, together, the source and the tag identify a piece of data, which should not be duplicated in a node. In each node, a piece of data is identified by the source node, the tag, and the lifetime of the request. If a piece of data is supposed to be copied to a node, which already has that particular piece of data, the copying is avoided, and the lifetime of the data is updated to reflect the latest request.

```
Calculate_Capacity ( )
begin
    for each node i
        max_capacity of i = curr_capacity of i = initial capacity of i
    for each request for data <s, tag> from s to d, with lifetime l and timestamp ts
        eliminate old data from nodes, according to ts
        update curr_capacity for the nodes which had data eliminated
        for each node k receiving a copy of the data
            if node k already has data <s, tag>
                update the lifetime for <s, tag> in node k
            else
                update curr_capacity of k
                if (curr_capacity of k > max_capacity of k)
                    max_capacity of k = curr_capacity of k
            end if
        end for
    end for
end
```

The input and output values for the algorithm are described in the sub-sections below.

3.1 Inputs

Network Topology
The network topology is read from a file. Each line in the file represents a node in the network, and is described by the node's neighbors and the associated cost between the node and each neighbor. Data centers can be identified by specifying, in a separate file, the tags and each tag's corresponding size.

Traffic Trace File
A synthetic trace for data requests represents the workload for a given topology. We use a traffic simulation tool, Flexible Optical Network Traffic Simulator (FONTS) [7], to generate a trace of data requests that simulates on-demand and advance-reservation requests with different stochastic characteristics. FONTS is based on a stochastic model that incorporates a variety of variables to model data transfer behavior of applications requiring sustained bandwidth. FONTS can be used to model bulk data transfer within the grid infrastructure, and thus is suitable for our use. Though the program allows the user to fine-tune numerous simulation parameters, we are

interested in configuring the request type (advanced-reservation or on-demand), source node, and destination node. The user simply generates a trace with FONTS, and that trace can be passed directly to the capacity planner for interpretation.

Replication Strategy
As the results in [8] show that only two of the six proposed replication/caching strategies demonstrate practical viability, for the moment, the capacity planner just considers these two replicating options: cascading with caching and fast spread.

3.2 Outputs

The storage-capacity planner produces as output a table summarizing storage consumption in the network. Specifically, the report shows the number of requests serviced and the initial, maximum, and current capacity at each node.

4 Experiment

Our storage-capacity planner enables a study of the effect of different data replication strategies on storage consumption. It is hypothesized that data networks replicating with fast-spread will consume more resources than networks using cascading with caching. In particular, as the network size increases, data sets in fast-spread networks are distributed more quickly and, therefore, the overall capacity to accommodate the rapid distribution will quickly outgrow that of cascading networks.

An experiment was designed to test this hypothesis. The experiment uses a topology typical of grids—the ring with chords. The topological variables in the experiment include: the size of the network, measured by the total number of nodes, and the degree of interconnectedness, measured by the number of chords. The number of nodes in the networks studied ranged from 2 to 20. The networks were also determined by the interconnectivity parameter, where 2- to 10-chord networks were simulated. The chords were randomly placed. A sample 10-node ring with 5 chords, as used in the experiment, is shown in Fig. 1.

As for the input traffic traces, advanced reservation requests can be generated with FONTS. Relevant configurable variables for the FONTS traces include a uniform distribution for the source and destination of a request and the number of switching nodes (to match the topology under study). The final and most important variable is the replication strategy used, which can be either fast-spread or cascading with caching. Results from the simulations can be used to quantify either the substantiation or contradiction of the hypothesis. Graphs should also be plotted to visualize potential trends from the collected data that, otherwise, might have been overlooked in the hypothesis.

The storage-capacity planner output is a table detailing the initial, current, and maximum storage resources consumed at each node in the network during the simulation. We aggregate the results into a succinct and meaningful mathematical value so that the simulated network configurations can be easily compared to one another. That mathematical value is obtained by averaging the maximum storage consumed, which provides a useful index for evaluating the overall distribution of maximum

resources utilized. The calculated indices are plotted as average capacity versus the number of nodes in the simulated network. The number of chords is also varied. A representative set of these experiments is shown in Fig. 2 to Fig. 6.

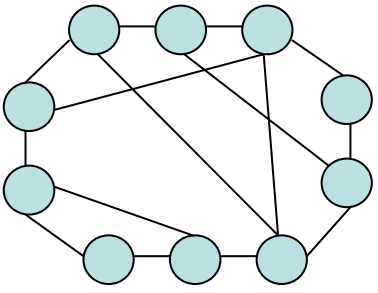


Fig. 1. 10-node ring with 5 chords

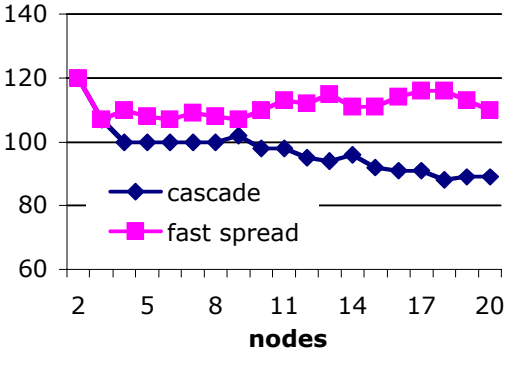


Fig. 2. Ring with no chords

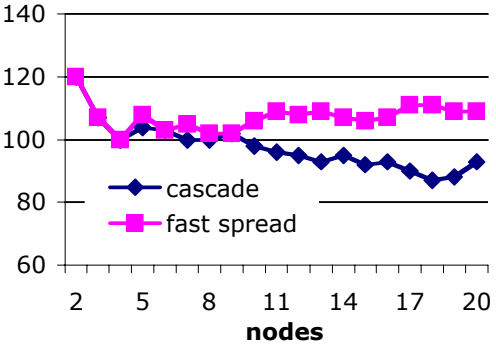


Fig. 3. Ring with 2 chords

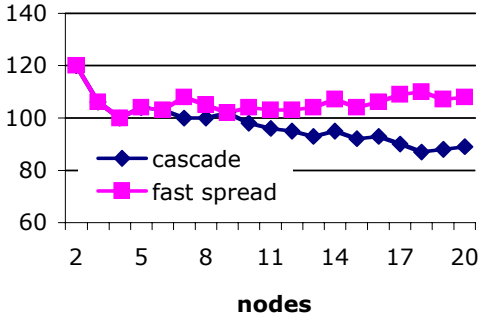


Fig. 4. Ring with 4 chords

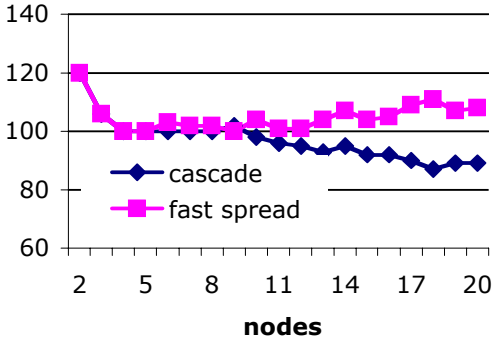


Fig. 5. Ring with 8 chords

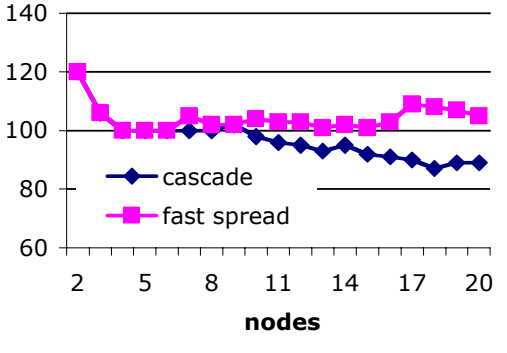


Fig. 6. Ring with 10 chords

The trends observed in the graphs confirm the hypothesis—that is, fast-spread replication on average consumes more storage resources than cascading with caching. With increasing network size, the storage utilization gap between the two replication strategies widens. The results correspond to the intuition that, if we more readily replicate,

then the overall storage spending increases accordingly. When there are more nodes in the network, since the fast-spread replication algorithm may distribute data to nearly all the nodes, the average storage usage shows a relatively uniform distribution. However, the more conservative cascading algorithm replicates only after a predetermined threshold number of requests have been reached. As network size increases, a smaller fraction of the nodes in the network will contain replicated data. Therefore, with an increasing network size, the average storage utilization shows a decreasing (somewhat linear) trend. The fast-spread strategy's somewhat constant average capacity, together with the cascading strategy's decreasing average capacity, explains the widening gap in resource utilization between the two replication strategies.

An unanticipated result is the overlapping capacity indices between the replication strategies when there are approximately five to nine nodes in the network. The observed trend sharpens with an increased interconnectivity. This finding provides valuable insight for data-grid designers, because it mitigates the often complicated tradeoff between time (data access speed) and space (storage resources). If the network is relatively small, such that there are no more than ten nodes, and some interconnectivity exists, then the tradeoff is unnecessary. Fast-spread data replication will consume on average the same amount of capacity as cascading while providing low access latency. However, as the number of nodes increases, the network designer must more carefully consider the time-and-space tradeoff. If access speed is critical, we must endure the increased storage consumption for low latency access. If the budget for storage resources is restricted, then use cascading replication to conserve capacity. There exists also demand for relatively low latency, but with a limited storage budget. Such a scenario requires an average capacity that lies somewhere between that of fast-spread and cascading, which may be achieved by a combination of the two replication strategies.

5 Conclusion

Data replication has been explored as a performance-tuning parameter for data grids, wherein two replication strategies (fast-spread and cascading with caching) have been identified to effectively reduce access latency or bandwidth consumption. A storage-capacity planner was developed to help data-grid designers better gauge the cost (in terms of storage resources) for fast-spread and cascading replication. An experiment was designed to estimate capacity consumption of the two replication strategies on ring networks. The results of the experiment coincided with the intuition that fast-spread networks consume more resources than cascading networks. The time-and-space tradeoff in network design was discussed, and a hypothesis was made that certain scenarios may require the use of both replication strategies in order to achieve good latency with moderate, or limited, storage resource funds. As future work, this feature can be incorporated into the storage-capacity planner, and a new experiment conducted to verify if a network combining both replication strategies would indeed yield the expected capacity consumption.

References

1. Hospodor, A.: Capacity Planning of Commercial and Scientific Grids with GridPlan. In: GlobusWORLD (2006)
2. Figueira, S., Kaushik, N., Naiksatam, S., Chiappari, S.A., Bhatnagar, N.: Advance Reservation of Lightpaths in Optical-Network Based Grids. In: Proceedings of the ICST/IEEE Gridnets (2004)
3. Foster, Kesselman, C., Tuecke, S.: The Anatomy of the Grid - Enabling Scalable Virtual Organizations. International Journal of High-Performance Computing Applications (2001)
4. GriPhyN - Grid Physics Network (2005), http://www.griphyn.org/
5. Ting, K., Liu, T., Tibbets, L., Figueira, S.: CapPlan - A Network Capacity Planning Tool for LambdaGrids. In: Proceedings of the IARIA/IEEE ICNS (2006)
6. Menasce, D.A., Almeida, V.A.: Capacity Planning for Web Services. Prentice Hall, Englewood Cliffs (2001)
7. Naiksatam, S., Figueira, S., Chiappari, S.A., Bhatnagar, N.: Analyzing the Advance Reservation of Lightpaths in Lambda-Grids. In: Proceedings of the IEEE/ACM CCGRID (2005)
8. Ranganthan, K., Foster, I.: Identifying Dynamic Replication Strategies for a High-Performance Data Grid. In: Proceedings of the International Workshop on Grid Computing (2002)
9. Stockinger, H., Samar, A., Allcock, B., Foster, I., Holtman, K., Tierney, B.: File and Object Replication in Data Grids. In: Proceedings of the 10th HPDC - International Symposium on High Performance Distributed Computing (2001)
10. The DataGrid Project, European Union (2005), http://eu-datagrid.web.cern.ch/eu-datagrid/
11. The Globus Data Grid Effort (2005), http://www.globus.org/toolkit/docs/2.4/datagrid/

Text Mining Grid Services for Multiple Environments

Antonio Anddre Serpa, Valeriana G. Roncero, Myrian C. A. Costa, and Nelson F.F. Ebecken

COPPE/Federal University of Rio de Janeiro,
P.O.Box 68516, 21945-970 Rio de Janeiro RJ, Brazil
Tel.: (+55) 21 25628081, Fax: (+55) 21 25628080
serpa@nacad.ufrj.br, valery@nacad.ufrj.br, myrian@nacad.ufrj.br, nelson@ntt.ufrj.br

Abstract. The objective of this paper is to describe the implementation of text mining grid services for Aîuri Project, which is a framework that includes a friendly user interface, data and text mining tasks, database access and a visualization tool integrated with various grid environments. The focus is the development and test of components for analysis and evaluation of unstructured data into distinct grid environments. These components are grid services for text mining processes using several approaches of execution, depending on which grid environment the user choose to submit his jobs. All components are open source and are freely available to the scientific community, providing access to existing services as well as encouraging the addition of new ones.

Keywords: Text Mining, Categorization, Grid Computing and Portal.

1 Introduction

Due to the continuous growth of the volume of available electronic data, automatic knowledge discovery techniques become necessary in order to manipulate huge amounts of data. Huge amounts of numerical data and countless pages of text are produced every day, in the academic or enterprise fields, documenting projects, actions or ideas. All the knowledge expressed in structured or unstructured form represents the most important property of an institution, either competitive advantage for companies or the availability of concepts and ideas for the academia. The techniques of text mining aims at extracting implicit knowledge in a collection of texts and documents [1].

Nowadays, the advances in education and research in the areas of text mining are leading to a torrent of new algorithms and methodologies for solving complex engineering and advanced sciences problems. Teaching those new algorithms and methods becomes itself a big challenge, if it is supposed that the use of programs with a friendly user interface and efficient visualization tools is necessary, even though this is not the main focus of the work and sometimes it is not present. Such difficulties can be minimized with the utilization of a well-defined environment that contains the previously mentioned facilities, leading to time savings on development, as well as keeping the focus on development and test of algorithms.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 576–587, 2008.

Grid computing is an infrastructure that integrates distributed and heterogeneous hardware and software resources, providing a virtual platform for computation and data management [2]. The growth of the environments of cooperative research enables the collaboration of researchers from geographically scattered locations. This scenario demands new dynamics and software that fulfills the modern needs of the researchers, such that development of new methodologies that integrate several development phases of new applications is possible.

However, the benefits brought by the use of computational grids can not be fully explored if they can not be easily accessed by an ordinary user. The user needs a friendly interface to interact with the grid environment. The contribution of this work is the construction of a friendly tool that allows the users to submit their tasks into three distinct environments, so that they can select the most suitable one to his task. This access is transparent to the user [16]. To test the tool two well-known text mining algorithms are used: Naïve Bayes and Linear Score, using the concepts of grid services.

2 Description of the Aîuri Project

The objective of Aîuri Project is the creation of a friendly user interface to provide easy access to at least one grid environment that is used to run, for instance, text mining algorithms. The web-based interface built in the Aîuri Project is, in fact, a framework that aggregates a friendly user interface and the tasks of text mining, using one or more grid environments. The focus of the project is the development of a high performance academic cooperative environment, which will be used for education and research in the areas of computational intelligence, analysis, evaluation and visualization of data via grid services that encapsulate the algorithms of data and text mining process. The software integrated in this environment is open source and available for the community, which will be capable of accessing built-in services and/or add new ones.

Grid computing infrastructure aggregates the collaboration feature, allowing the utilization and/or incorporation of new strategies for the research of new algorithms and different approaches for the solution of advanced engineering and science problems.

This paper describes the implementation of a text mining grid service for the Aîuri Project.

3 Text Mining Tasks on the Aîuri Environment

Text Mining, also known as Knowledge Discovery form textual databases [3], refers to the non trivial extraction of implicit, previously unknown, and potentially useful information from large amounts of textual data, such as documents and unstructured data. Text Mining describes an assembly of processes with algorithms and efficient techniques that allow the manipulation of texts. Different algorithms can be used depending on the discovery goal.

The main phases of text mining process are text preparation, text mining pattern discovery and post-processing or interpretation and evaluation; this is shown in figure 1.

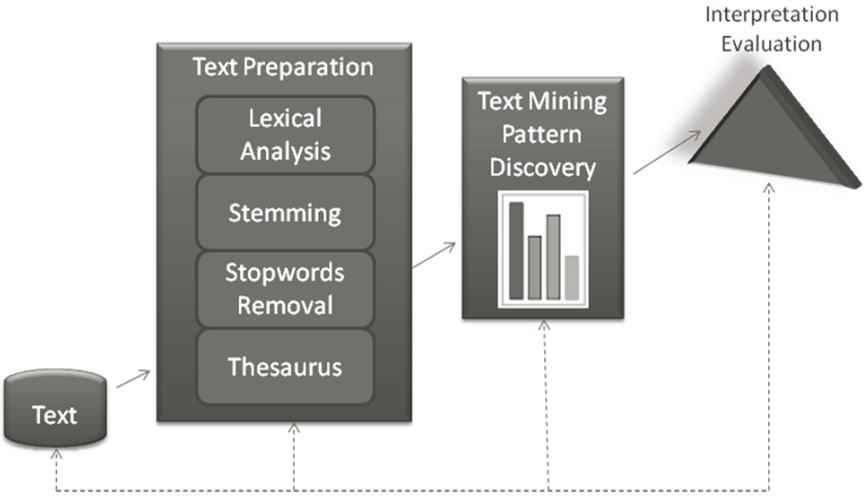


Fig. 1. Scheme of Text Mining Process, showing its main phases

3.1 Text Preparation

The text preparation phase consists of the selection of terms that better describe document contents, neglecting any unimportant information. This type of selection improves performance and classification effectiveness. The activities that must be performed are: lexical analysis that identifies each term as a character sequence; morphological analysis or stemming that reduces each term to its radical, made by a stemmer; stop words removal that consists of removing terms with no special meaning from the texts, like prepositions, articles and conjunctions; utilization of a thesaurus in order to replace different terms with a key term that has the semantic meaning.

The use of these techniques results in a collection of representations of words of a document, which is mapped into a term-document table.

3.2 Text Mining Pattern Discovery

Among the diverse approaches of analysis for the extraction of knowledge, categorization tasks are intended to automatically classify documents related to a collection of previously defined categories [4]. This task can be used, for example, to insert a new document in a collection divided into categories. To achieve this goal the categories must be represented by terms or a set of terms that bear the meaning of the category concept. The categorization technique is defined as the process of finding a model that describes a category. Given a collection of labeled records, each of them containing a set of features and the category, the model for a category is a function of the values of the features. Usually, the data set is divided into training and test sets, where the training set is used to build the model and test set is used to validate it and determine its accuracy. The goal of this process is to assign categories to previously unseen records as accurately as possible.

3.3 Post-processing

The post-processing task consists of the validation, visualization and evaluation of the obtained patterns from the expert point of view. The evaluation accuracy and the visualization tools are especially important to achieve the most useful and relevant conclusions. This phase is not treated in the present paper, because post-processing is subject of research and development in future projects.

4 The Aîuri Portal

The Aîuri portal is a web interface that allows users to request execution of text mining tasks in three distinct environments. Due to the particular features of each type of task, the application behavior varies depending on how the user chooses the task parameters. The access to the portal is controlled by a user/password authentication method. All the services are available to all users. This is summarized in table 1.

Table 1. Portal Functions

Function	Description
Upload Certificate	It carries out the load of the certificate.
Upload File	It carries out the load of the files of the user.
Training set XML	Generation of the file with the training set.
Test set XML	Generation of the file with the test set.
Make Stemmer	Generation of the file with stems.
Bayesian categorization	Naïve Bayes categorizer.
Linear categorization	Linear Score categorizer.

The portal encapsulates the structure of the Aîuri project. The Aîuri project has three main components: a portal, which is the interface between user and the services and the grid services, which implement the tasks of text mining and maintain all the files uploaded to the environment.

Web-based Grid computing portals are effective tools for providing users with simple, intuitive interfaces for accessing grid information and its resources [5]. The software used to build grid portals interacts with the middleware running on the resources. The portal software must be compatible with common Web servers and browsers/clients. Grid portals make the distributed heterogeneous computing and data grid environments more accessible to grid users by using common Web and UI (User Interfaces) conventions.

The processes performed by the portal are shown in the figure 2. The first step is the upload of the data necessary to perform the mining task. For the execution of a text mining task all the text files which are part of the knowledge base must be uploaded. Each user has a private knowledge base in an exclusive area. Once the files are uploaded, the next step is to create the training and test sets, by converting them to the XML format. These sets will be stored in the user area too. In the grid environment these uploaded and converted files can be stored in the environment.

After that, the user can choose the parameters of the text mining task, enabling the portal to generate an object with the states of the parameters. Among other parameters, the user can specify the name of the dictionary, the amount of words and the key term for the categorization task.

At this moment the portal queries the available resources for the grid execution and lets the user choose the appropriate resource. The grid information service provides the states and workload of the grid resources. For the submission of the text mining task for execution, the portal contacts the job management service of the grid and waits for the task results.

The results are sent back to the portal and visualized by the user.

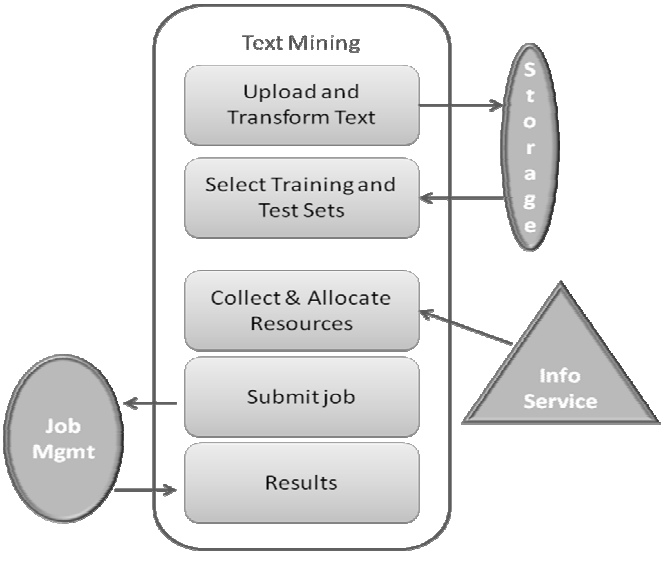


Fig. 2. Aîuri portal processes

4.1 Text Mining Grid Service

The first application implemented is text mining categorization. The application was developed in Java and is performed by the Naïve Bayes and Linear Score algorithms. The probabilistic Naïve Bayes (NB) classifier has been widely used with good performance for document classification. It is based on the Bayes' Theorem [6,7]. The basic idea is to join the key words in categories to estimate the probabilities of the categories of a new document. The algorithm computes the a posteriori probabilities of a document to belong to distinct classes and assigns it to the class with larger a posteriori probability. The a posteriori probability is computed using the Bayes' rule and the test set is assigned to the class with the largest a posteriori probability. The naïve part of the NB algorithm is the independence assumption of the characteristics of the word, that is, it is assumed that the effect of the characteristics of the word of which conditional probability is associated to a category is independent of the characteristics of the other words of that category. Several experiments have been performed

with the NB algorithm [8], presenting satisfactory results for pattern classification. However, other methods are also used in the text mining area. The NB algorithm presents several advantages over other techniques. It is quite simple and easy to implement. Additionally, no learning process is required, because the probabilities are estimated based on the frequency of the terms. Moreover, the classification process is efficient, since the characteristics are independent of each other. On the other hand, the NB algorithm has some drawbacks. It requires many probabilities to be known *a priori*, and the computing cost grows linearly, depending on the quantity of existing words and characteristics.

The linear score classifier is based on linear methods, which are a classical approach for the solution of prediction problems. The Bayesian method, used in this work, can be viewed as a special case of the linear method, but without problems with redundant attributes, since it performs better when the number of attributes is small, which is not the case when dictionaries with thousands of words are created. The linear score method [11] sets a positive score to the classes identified as positive and a negative one to the classes identified as negative, such that for every word that appears in a document its corresponding weight is determined. These weights must be summed to compute the score of the document. An advantage of the linear approach is the simplicity of the construction of the model, provided that a set of significant terms of the document is chosen and the learning algorithm is capable of determining the weight of each term created.

5 Grid Environments

A grid is an internet-connected computing environment in which computing and data resources are geographically distributed over different administrative domains, often with separate policies for security and use of resources [9]. Two distinct computational grid environments are used in this work: the NACAD Grid, installed at the NACAD laboratory, and the EELA Grid.

5.1 NACAD Grid Environment

The NACAD Grid uses Globus GT4 [10] as grid middleware. In order to integrate the framework to the GT4 infrastructure, some entities (classes) were created to allow the integration of the system with this new environment. GT4 is a grid middleware based on grid services. Grid services is a technology based on the concepts and technologies of grids and web services and can be defined as a web service that delivers a set of interfaces that follows specific conventions. This technology was originated from the necessity to integrate services through virtual, heterogeneous and dynamic organizations, composed of distinct resources, whether within the same organization or by resource sharing. The structure of the NACAD grid is shown in figure 3.

5.2 EELA Grid Envrionment

The second integration was carried out within the EELA (E-Infrastructure Shared Between Europe and Latin America) Project. The objective of the EELA Project is to establish a human collaboration network to share an infrastructure to support test and

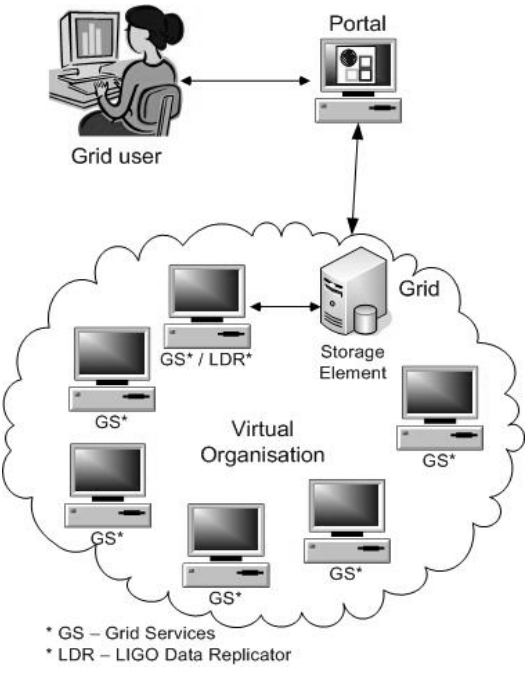


Fig. 3. Components of the Aîuri project

development of advanced applications. EELA-2 is the second phase of EELA Project. The integration to this environment required the implementation of the following activities: (i) Use of AMGA (ARDA Metadata Grid Application): AMGA [12] is a metadata service for computational grids. It can be viewed as a data base access service for grid applications, which enables jobs running on the grid to access the data base, providing authentication, as well as a layer that hides from the user the technical details of distinct data bases, providing the user a unique method of access to all data bases in the environment. In fact, AMGA is a service that functions between the SGDB and the client application. AMGA is used to create a structure that validates the user at the moment he logs into the environment; (ii) Use of GSAF (Grid Storage Access Framework): GSAF [13] is a framework that encapsulates the access method to data by providing API's that enable access to AMGA, to the files catalogue and storage elements. The structure of AMGA and use of GSAF are shown in figures 5a and 5b; (iii) Job submission: to submit jobs to the EELA grid the API LCG (LHC Computing Grid) is used. This API provides a number of functionalities to access this environment. To submit jobs using this API a file that describes the properties of the jobs must be created. The Job Description Language [14] (JDL) is a language intended to create such description. The class implemented to perform the submission activities is an example of a JDL file is shown in figure 6.

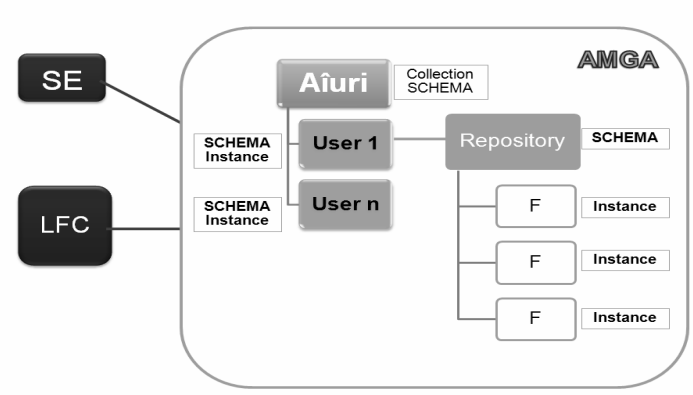


Fig. 5a. Schematic example of the AMGA structure for Aîuri Project

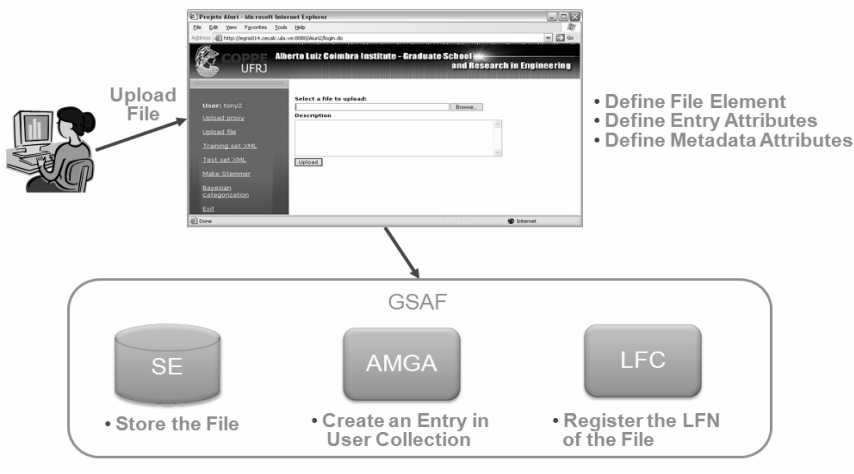


Fig. 5b. Use of GSAF structure

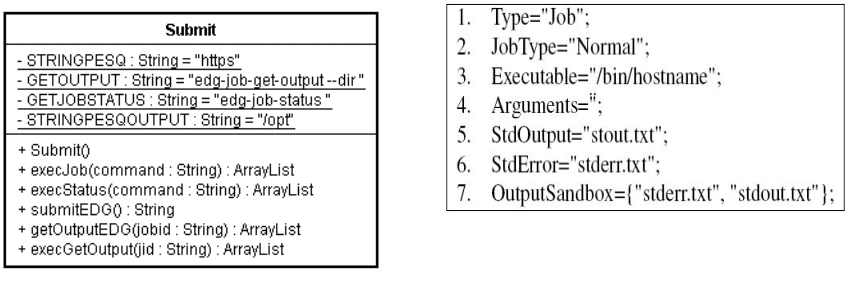


Fig. 6. Example of the GSAF architecture

It is necessary to remark that the work philosophy of the Aîuri Portal is to provide grid services to its users. In a preliminary approach, using the grid structure available at NACAD, all the grid services implemented are based on web services technology. In a second approach, using the EELA grid, the grid services are not based on web services, but its services are also enabled for the users by the portal.

6 Experiments

In order to evaluate the quality of the results obtained and, specially, to validate the environments, two experiments are presented. Two sets of texts are used, taken from CETENFolha [15], which are previously classified by Computational Process of Portuguese Project. This previous classification is used to create a categorization model and later, to check the categorized results. The tests were carried out with and without balancing. These tests were performed based on four distinct approaches: (i) stems and stop words were not used; (ii) stop words were used; (iii) stems and stop words were both used; (iv) only stems were used. The graphics are used to display the results of the models, with their corresponding f-measures and time processing for the best models created using a local computer and the NACAD Grid. The structure of the tests is shown in table 2.

Table 2. Structure of the experiments

Not Balanced			**Balanced**		
Topic	**Trainning**	**Test**	**Topic**	**Trainning**	**Test**
Brazil	2483	100	Brazil	1250	250
Money	2124	100	Money	1250	250
Sport	2594	100	Sport	1250	250
World	1565	100	World	1250	250

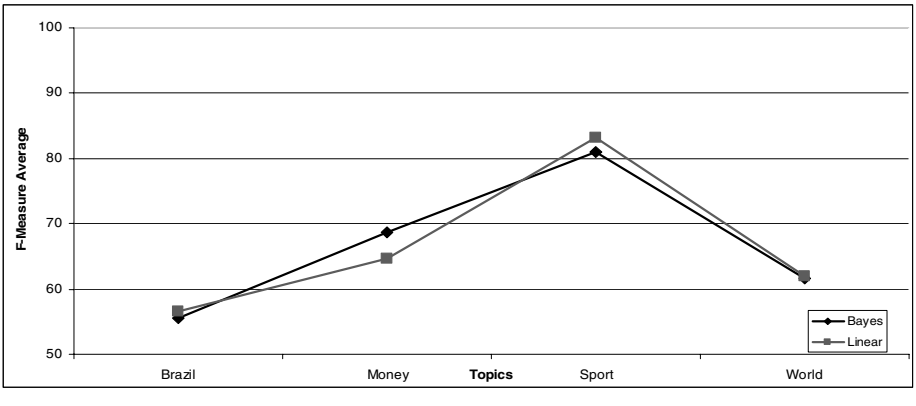


Fig. 7a. Comparison of the results computed by the bayesian and linear score categorizers (not balanced)

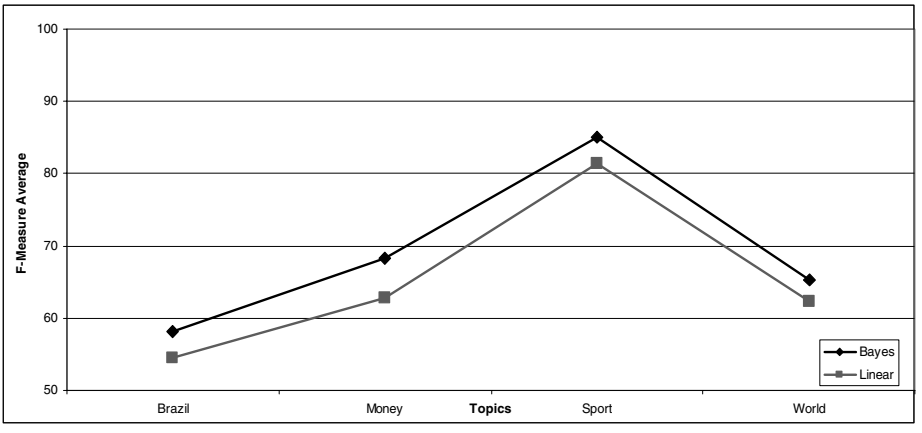


Fig. 7b. Comparison of the results computed by the bayesian and linear score categorizers (balanced)

The experiments show that the best classification results were obtained with the class *Sport*. This is so, certainly, because the vocabulary in this class is quite distinct from the vocabulary of the remaining classes. During the experiments, it was observed that the use of stems, in average, caused an improvement of about 2% in the generated models. Both categorizers show similar results, as shown in figures 7a and 7b. However, the Bayesian categorizer, in average, performed faster, and obtained classification models slight better than those obtained by the linear score categorizer.

7 Conclusion

We discuss the use of the Aîuri Portal for the execution of text mining grid services processed in grids environments.

The Aîuri Portal provides a very effective means of submission of algorithms to grid environments, since its functionalities encapsulate several tasks that, without the portal, would be performed by the user, making the use of computational grids accessible to ordinary users.

An additional advantage of our portal is its versatility, since two grid environments and the local machine are available to the user, and more environments can be incorporated to it.

The experiments show very good text mining processing times on a grid environment compared to local execution, encouraging the continuity of the development.

The results obtained with the Bayesian algorithm were slightly better than those obtained with the linear score algorithm. The algorithms were tested using global dictionaries, which boosts the quality of the models when compared with the use of local dictionaries.

Acknowledgements

The authors would like to thank the High Performance Computing Center (NACAD) at the Graduate School and Research in Engineering (COPPE), Federal University of Rio de Janeiro (UFRJ) for providing the computational resources for this research.

The authors would like also to thank to EELA Project for supporting the gridification of text mining grid service in the context of EELA in EGRIS-2 (Second EELA Grid School).

The Aîuri Project is an application supported by the EELA-2 project, funded by the European Commission under the Grant Agreement #223797, where it would be developed and deployed.

References

[1] Lopes, M.C., Costa, M.C.A., Ebecken, N.F.F.: Text Mining. In: Rezende, S.O (Org.). Intelligent Systems: Foundations and Applications (in Portuguese). Editora Manole Ltda (2002)

[2] Berman, F., Fox, G., Hey, T.: The Grid: past, present, future. In: Berman, F., Fox, G., Hey, T. (eds.) Grid Computing: Making the Global Infrastructure a Reality. John Wiley & Sons, Chichester (2003)

[3] Feldman, R., Dagan, I.: Knowledge discovery in textual databases (KDT). In: Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD 1995), Montreal, Canada. AAAI Press, Menlo Park (1995)

[4] Sebastiani, F.: A Tutorial on Automated Text Categorization. In: Proceedings of THAI 1999, European Symposium on Telematics, Hypermedia and Artificial Intelligence, Italy (1999)

[5] Thomas, M., Boisseau, J.: Building Grid computing portals: The NPACI Grid portal toolkit. In: Berman, F., Fox, G., Hey, T. (eds.) Grid Computing: Making the Global Infrastructure a Reality. John Wiley & Sons, Chichester (2003)

[6] Tzeras, K., Hartman, S.: Automatic indexing based on bayesian inference networks. In: Proceedings 16th ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1993), pp. 22–34 (1993)

[7] Lewis, D.D., Ringuette, M.: Comparison of two learning algorithms for text categorization. In: Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval (SDAIR 1994) (1994)

[8] Lacerda, W.S., Braga, A.P.: Experiments of a Pattern Classifier based on Naive Bayes Rule (in Portuguese). In: INFOCOMP - Revista de Computação da UFLA, Lavras, vol. 1(3), pp. 30–35 (2004)

[9] Qi, L., Jin, H., Foster, I., Gawor, J.: HAND: Highly Available Dynamic Deployment Infrastructure for Globus Toolkit 4. Document, `http://www.globus.org/alliance/publications/papers/HAND-Submitted.pdf`

[10] Foster, I.: Globus Toolkit Version 4: Software for Service-Oriented Systems. In: Jin, H., Reed, D., Jiang, W. (eds.) NPC 2005. LNCS, vol. 3779, pp. 2–13. Springer, Heidelberg (2005)

[11] Weiss, S.M., Indurkhya, N., Zhang, T., Damerau, F.J.: Text Mining: Predictive Methods for Analyzing Unstructured Information, p. 237. Springer Science+Business Media, New York (2005)
[12] Koblitz, B., Santos, N.: AMGA User's and Administrator's Manual (November 2006)
[13] Scifo, S.: GSAF-Grid Storage Access Framework (June 2007)
[14] Pacini, F.: Job Description Language – How To (December 2001)
[15] CentenFolha – Conjunto de Textos para Mineração de Textos URL, http://acdc.linguateca.pt/cetenfolha/
[16] Serpa, A.A.: Aîuri: A Web Portal for Text Mining integrated to Computational Grids, M.Sc. Dissertation, COPPE/UFRJ (2007)

Using Stemming Algorithms on a Grid Environment

Valeriana G. Roncero, Myrian C.A. Costa, and Nelson F.F. Ebecken

COPPE/Federal University of Rio de Janeiro
P.O. Box 68516, 21945-970, Rio de Janeiro, RJ, Brazil.
{valery,myrian}@nacad.ufrj.br, nelson@ntt.ufrj.br

Abstract. Stemming algorithms are commonly used in Information Retrieval with the goal of reducing the number of the words which are in the same morphological variant in a common representation. Stemming analysis is one of the tasks of the pre-processing phase on text mining that consumes a lot of time. This study proposes a model of distributed stemming analysis on a grid environment to reduce the stemming processing time; this speeds up the text preparation. This model can be integrated into grid-based text mining tool, helping to improve the overall performance of the text mining process.

Keywords: Grid environment, distributed computing, text mining, stemming analysis.

1 Introduction

The enormous amount of information stored in unstructured texts cannot simply be processed by computers which typically handle text as simple sequences of character strings. Text mining is the process of extracting interesting information and knowledge from unstructured text. It runs several processes, such as document collection, pre-processing and preparation, pattern discovery and evaluation and interpretation of the results.

Text mining techniques gained considerable importance as a technology to retrieve data from huge amount of digitally stored documents [1]. In order to extract useful patterns [2] pre-processing tasks and algorithms are required. Stemming analysis, which is used in this paper, is one of the pre-processing tasks of text mining.

Due to the large quantity of documents, pre-processing tasks are computationally intensive. In order to reduce the time spent in pre-processing we distribute the stemming analysis on a grid environment. This environment is a geographically distributed computation infrastructure composed of a set of heterogeneous resources.

2 Text Mining

Text mining is a relatively new practice derived from Information Retrieval (IR) [3, 4] and Natural Language Processing (NLP) [5]. The strict definition of text mining includes only the methods capable of discovering new information that is not obvious or easy to find out in a document collection, i.e., reports, historical documents, e-mails, spreadsheets, papers and others.

J.M.L.M. Palma et al. (Eds.): VECPAR 2008, LNCS 5336, pp. 588–593, 2008.

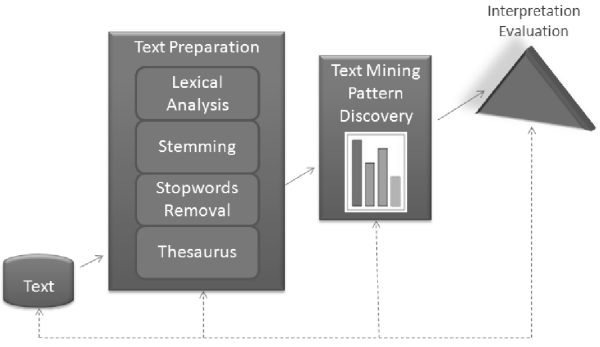


Fig. 1. Summary of the text mining phases

Text mining executes several processes, each one consisting of multiple phases, which transform or organize an amount of documents in a systematized structure. These phases enable the use further processed documents, in an efficient and intelligent way. The processes that compose the text mining can be visualized in Figure 1 that is a summarized version of the figure model from [6] on page 6.

The text mining processes are divided into the following phases:

1. Document collection: consists of the definition of the documents set from which knowledge must be extracted.
2. Pre-processing and preparation: consists of a set of actions that transform the set of documents in natural language into a list of useful terms. Then from these terms will be identified and selected the relevant terms.
3. Text Mining Pattern Discovery: consists of the application of machine learning techniques to identify patterns that can classify or cluster the documents in the collection.
4. Evaluation and interpretation of the results: consists of the results analysis.

The pre-processing phase in text mining is essential and usually time consuming. As texts are originally non-structured, some steps are required to represent them in a format which is compatible with knowledge extraction methods and tools.

2.1 The Stemming Process

The stemming process is an important pre-processing task before indexing input documents for text mining. The term stemming refers to the reduction of words to their roots so that, different grammatical forms or declinations of verbs are identified and indexed (counted) as the same word. For example, stemming will ensure that both “takes” and “take” will be recognized by the program as the same word [7]. In most cases, morphological variants of words have similar semantic interpretations and can be considered as equivalent for the purpose of Information Retrieval applications. For this reason, a number of so-called stemming Algorithms, or stemmers, have been developed, which attempt to reduce a word to its root form.

Lovins [8] described the first stemmer developed specifically for Information Retrieval applications and introduced the idea of stemming based on a dictionary of common suffixes. This algorithm stimulated the development of many subsequent algorithms [9, 10] and, more generally, the use of stemming as a general tool in the Information Retrieval area [10, 11, 12, 13, 14].

In this model two well-known stemmer algorithms will be implemented: Porter stemmer and Paice/Husk stemmer. The Porter stemming algorithm [15] is a process in which is removed the commoner morphological and suffixes from words. Its main use is as part of a term normalization process that is usually done when setting up Information Retrieval systems [16]. The Paice/Husk stemmer [17] is iterative and uses a single table of rules. Each rule may specify the removal or replacement of an ending. The rules are grouped into sections corresponding to the final letter of the suffix; this meas that the rule table is accessed quickly by looking up the final letter of the current word or truncated word.

3 Grid Environment

A grid is a geographically distributed computation infrastructure composed of a set of heterogeneous machines, often with separate policies for security and resource use [18], that users can access via a single interface. Grids therefore, provide a common resource-access technology and operational services across widely distributed virtual organizations composed of institutions or individuals that share resources. Today grids can be used as effective infrastructures for distributed high-performance computing and data processing [19]. Figure 2 shows the Grid NACAD infrastructure that will be used in this work.

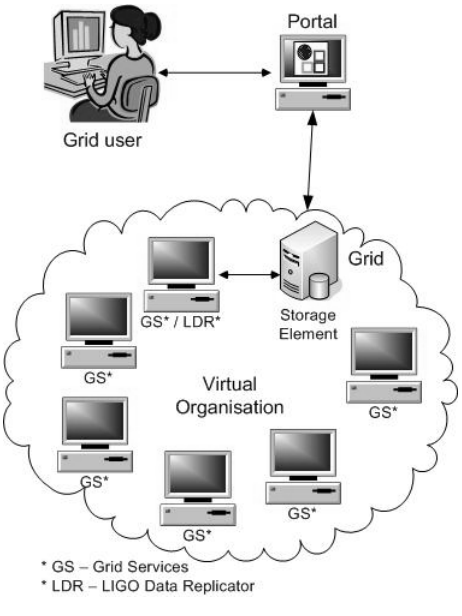


Fig. 2. The Grid NACAD infrastructure

In this work we use the Globus Toolkit 4 (GT4) [20], which is a widely used middleware in scientific and data-intensive grid applications, and is becoming standard for implementing grid systems. The toolkit addresses security, information discovery, resource and data management, communication, fault-detection, and portability issues. Nowadays, Globus and the other grid tools are used in many projects worldwide. Although there is most of these projects are in scientific and technical computing fields, a growing number of grid projects in education, industry, and commerce being implemented.

4 Distributed Stemming Analysis on a Grid

The pre-processing phase is very time consuming, particulary the stemming task. This is so because of the large number of words that a document collection contains. To reduce the time spent for stemming, we distribute the documents on a grid environment to process the stemming simultaneously in the grid nodes.

The Globus Toolkit provides a number of components for performing data management. Data management tools (GridFTP, RFT, RLS) are concerned with the location, transfer, and management of distributed data [21]. GridFTP protocol provides a secure way to transfer data in a grid. RFT (Reliable File Transfer) is a Web Services Resource Framework (WSRF) [22] compliant web service for managing multiple data transfers. The Replica Location Service (RLS) [23] maintains and provides access to mapping information from logical names for data items onto target names. These target names may represent physical locations of data items, or an entry in the RLS may map to another level of logical naming for the data item. The RLS is intended to be one of a set of services for providing data replication management in grids. In additional of this components, the LIGO Data Replicator (LDR) [24, 25] will be used. LDR is a collection of some components provided by the Globus project with some extra logic to pull the components together. This minimum collection of components is necessary for fast, efficient, robust, and secure replication of data. The Globus components included are: GridFTP, Globus Replica Location Service (RLS) and a metadata service developed by the LDR team but based on a prototype Globus Metadata Catalog Service (MCS) [26] for organizing useful information about the data files, especially as it pertains to when and where the data should be replicated.

Figure 3 shows the distributed stemming analysis on a grid model. The grid user owns a grid certificate, which provides him the grid credentials [27] to log into the grid and submit jobs to it, which is done by means of a Portal, accessible from the user's workstation. After logged in, the user can access his documents or public documents that are stored in the grid. He submits to the Portal information about the documents that will be analyzed (1). The Portal uses the LDR queries to find out whether there is a local copy of the documents, if not, RLS tells the Portal where the documents are in the grid (2). Then the LDR system generates a request to copy the documents to the local storage system and registers the new copy in the local RLS server. The grid nodes receive from the Portal the phases to run the stemming task (3) and using the RFT service it copies the replicas of the documents from the storage to the grid nodes (4). When the stemming task is concluded, the Portal collects all sets of documents from

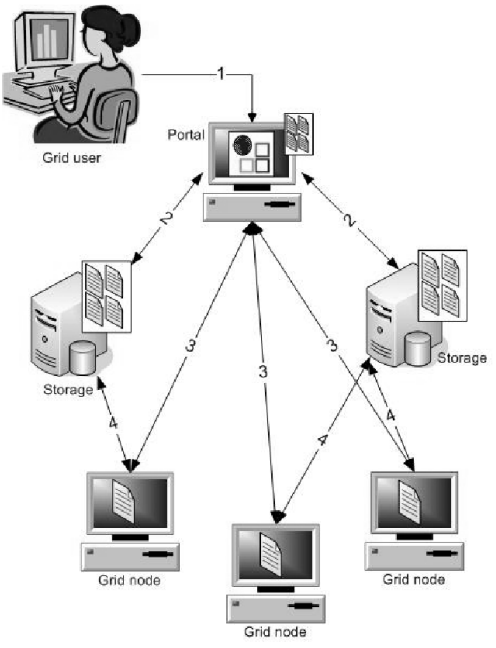


Fig. 3. Distributed stemming analysis model

each node and returns the result of the stemming to the user, who stores the documents in his grid account area.

5 Summary

In this paper we presented a distributed stemming analysis model. This model focuses on reducing the stemming task processing time, using a grid environment to distribute the documents to speed up the stemming task within a group of documents. The next step is to develop this model and integrate it to a text mining system through a grid service using the Globus Toolkit middleware in the Grid NACAD.

Acknowledgments. The authors would like to thank the High Performance Computing Center (NACAD) at the Graduate School and Research in Engineering (COPPE), Federal University of Rio de Janeiro for providing the computational resources for this research and The National Research Council of Brazil (CNPq) for financial support.

References

1. Hearst, M.A.: Untangling text data mining. In: Proceedings of the 37^{th} Annual Meeting on Computational Linguistics, pp. 3–10. Association for Computational Linguistics (1999)
2. Konchady, M.: Text Mining Application Programming. Charles River Media (2006)

3. Salton, G., McGill, M.J.: Introduction to Modern Information Retrieval. McGraw-Hill Book Company, New York (1983)
4. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. ACM Press Books, New York (1999)
5. Kao, A., Poteet, S.R.: Natural Language Processing and Text Mining. Springer, Heidelberg (2007)
6. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco (2001)
7. Manning, C.D., Schuetze, H.: Foundations of Statistical Natural Language Processing. MIT Press, Cambridge (1999)
8. Lovins, J.B.: Development of a stemming algorithm. Mechanical Translation and Computational Linguistics 11(1/2), 22–31 (1968)
9. Lennon, M., Peirce, D.S., Tarry, B.D.: Development of a stemming algorithm. Mechanical Translation and Computational Linguistics 3(4), 177–183 (1981)
10. Porter, M.F., Tait, J.I.: Charting a new course: Natural language processing and information retrieval. In: Essays in Honour of Karen Spärck Jones. pp. 39–68. Springer, Heidelberg (2005)
11. Frakes, W.B., Fox, C.J.: Strength and similarity of affix removal stemming algorithms. In: ACM SIGIR Forum., vol. 37, pp. 26–30 (2003)
12. Harman, D.: How effective is suffixing? Journal of the American Society for Information Science 42(1), 7–15 (1991)
13. Hull, D.A.: Stemming algorithms: a case study for detailed evaluation. Journal of the American Society for Information Science 47(1), 70–84 (1996)
14. Krovetz, B.: Viewing morphology as an inference process. Artificial Intelligence 118(1/2), 277–294 (2000)
15. Porter, M.F.: An algorithm for suffix stripping. Program (July 1980)
16. Porter, M.F.: The Porter stemming algorithm, `http://tartarus.org/~martin/PorterStemmer/index.html`
17. Paice, C.D.: Another stemmer. SIGIR Forum. 24(3), 56–61 (1990)
18. Qi, L., Jin, H., Foster, I., Gawor, J.: Hand: Highly available dynamic deployment infrastructure for globus toolkit 4, `http://www.globus.org/alliance/~publications/papers/HANDSubmitted.pdf`
19. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. Intl. J. Supercomputer Applications 15(3), 20 (2001)
20. The Globus Toolkit, `http://www.globus.org/toolkit/`
21. GT4 Data Management, `http://www.globus.org/toolkit/docs/4.0/data/`
22. The WS-Resource Framework, `http://www.globus.org/wsrf/`
23. Replica Location Service, `http://www.globus.org/toolkit/data/rls/`
24. LIGO Scientific Collaboration Research Group: Ligo Data Replicator., `http://www.lsc-group.phys.uwm.edu/LDR/`
25. Chervenak, A., Schuler, R., Kesselman, C., Koranda, S., Moe, B.: Wide area data replication for scientific collaborations. In: Proceedings of 6^{th} IEEE/ACM International Workshop on Grid Computing (Grid 2005) (November 2005)
26. Metadata Catalog Service, `http://www.globus.org/grid_software/data/mcs.php`
27. GT 4.0: Security: Pre-Web Services Authentication and Authorization, `http://www.globus.org/toolkit/docs/4.0/security/prewsaa/`

Author Index