Roman Wyrzykowski
Jack Dongarra
Konrad Karczewski
Jerzy Wasniewski (Eds.)

LNCS 6068

Parallel Processing and Applied Mathematics

8th International Conference, PPAM 2009
Wroclaw, Poland, September 2009
Revised Selected Papers, Part II

Springer

Lecture Notes in Computer Science 6068

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

Roman Wyrzykowski Jack Dongarra
Konrad Karczewski Jerzy Wasniewski (Eds.)

Parallel Processing and Applied Mathematics

8th International Conference, PPAM 2009
Wroclaw, Poland, September 13-16, 2009
Revised Selected Papers, Part II

Springer

Volume Editors

Roman Wyrzykowski
Konrad Karczewski
Czestochowa University of Technology
Institute of Computational and Information Sciences, Poland
E-mail:{roman, xeno}@icis.pcz.pl

Jack Dongarra
University of Tennessee, Department of Electrical Engineering
and Computer Science, Knoxville, TN 37996-3450, USA
E-mail: dongarra@cs.utk.edu

Jerzy Wasniewski
Technical University of Denmark, Department of Informatics
and Mathematical Modeling, 2800 Kongens Lyngby, Denmark
E-mail: jw@imm.dtu.dk

Library of Congress Control Number: 2010930224

CR Subject Classification (1998): D.2, H.4, D.4, C.2.4, D.1.3, F.2

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

ISSN 0302-9743
ISBN-10 3-642-14402-0 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-14402-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

We are pleased to present the proceedings of the 8th International Conference on Parallel Processing and Applied Mathematics – PPAM 2009, which was held in Wrocław, Poland, September 13–16, 2009. It was organized by the Department of Computer and Information Sciences of the Częstochowa University of Technology, with the help of the Wrocław University of Technology, Faculty of Computer Science and Management. The main organizer was Roman Wyrzykowski.

PPAM is a biennial conference. Seven previous events have been held in different places in Poland since 1994. The proceedings of the last four conferences have been published by Springer in the *Lecture Notes in Computer Science* series (Nałęczów, 2001, vol.2328; Częstochowa, 2003, vol.3019; Poznań, 2005, vol.3911; Gdańsk, 2007, vol. 4967).

The PPAM conferences have become an international forum for exchanging ideas between researchers involved in parallel and distributed computing, including theory and applications, as well as applied and computational mathematics. The focus of PPAM 2009 was on models, algorithms, and software tools which facilitate efficient and convenient utilization of modern parallel and distributed computing architectures, as well as on large-scale applications.

This meeting gathered more than 210 participants from 32 countries. A strict refereeing process resulted in the acceptance of 129 contributed presentations, while approximately 46% of the submissions were rejected. Regular tracks of the conference covered such important fields of parallel/distributed/grid computing and applied mathematics as:

- Parallel/distributed architectures and mobile computing
- Numerical algorithms and parallel numerics
- Parallel and distributed non-numerical algorithms
- Tools and environments for parallel/distributed/grid computing
- Applications of parallel/distributed computing
- Applied mathematics and neural networks

Plenary and Invited Speakers

The plenary and invited talks were presented by:

- Srinivas Aluru from the Iowa State University (USA)
- Dominik Behr from AMD (USA)
- Ewa Deelman from the University of Southern California (USA)
- Jack Dongarra from the University of Tennessee and Oak Ridge National Laboratory (USA)
- Iain Duff from the Rutherford Appleton Laboratory (UK)
- Anne C. Elster from NTNU, Trondheim (Norway)

- Wolfgang Gentzsch from the DEISA Project
- Michael Gschwind from the IBM T.J. Watson Research Center (USA)
- Fred Gustavson from the IBM T.J. Watson Research Center (USA)
- Simon Holland from Intel (UK)
- Vladik Kreinovich from the University of Texas at El Paso (USA)
- Magnus Peterson from the Synective Labs (Sweden)
- Armin Seyfried from the Juelich Supercomputing Centre (Germany)
- Bolesław Szymański from the Rensselaer Polytechnic Institute (USA)
- Jerzy Waśniewski from the Technical University of Denmark (Denmark)

Workshops and Minisymposia

Important and integral parts of the PPAM 2009 conference were the workshops:

- Minisymposium on GPU Computing organized by José R. Herrero from the Universitat Politecnica de Catalunya (Spain), Enrique S. Quintana-Ortí from the Universitat Jaime I (Spain), and Robert Strzodka from the Max-Planck-Institut für Informatik (Germany)
- The Second Minisymposium on Cell/B.E. Technologies organized by Roman Wyrzykowski from the Częstochowa University of Technology (Poland), and David A. Bader from the Georgia Institute of Technology (USA)
- Workshop on Memory Issues on Multi- and Manycore Platforms organized by Michael Bader and Carsten Trinitis from the TU München (Germany)
- Workshop on Novel Data Formats and Algorithms for High-Performance Computing organized by Fred Gustavson from the IBM T.J. Watson Research Center (USA), and Jerzy Waśniewski from the Technical University of Denmark (Denmark)
- Workshop on Scheduling for Parallel Computing - SPC 2009 organized by Maciej Drozdowski from the Poznań University of Technology (Poland)
- The Third Workshop on Language-Based Parallel Programming Models - WLPP 2009 organized by Ami Marowka from the Shenkar College of Engineering and Design in Ramat-Gan (Israel)
- The Second Workshop on Performance Evaluation of Parallel Applications on Large-Scale Systems organized by Jan Kwiatkowski, Dariusz Konieczny and Marcin Pawlik from the Wrocław University of Technology (Poland)
- The 4th Grid Application and Middleware Workshop - GAMW 2009 organized by Ewa Deelman from the University of Southern California (USA), and Norbert Meyer from the Poznań Supercomputing and Networking Center (Poland)
- The 4th Workshop on Large Scale Computations on Grids - LaSCoG 2009 organized by Marcin Paprzycki from IBS PAN and SWPS in Warsaw (Poland), and Dana Petcu from the Western University of Timisoara (Romania)
- Workshop on Parallel Computational Biology - PBC 2009 organized by David A. Bader from the Georgia Institute of Technology in Atlanta (USA), Denis Trystram from ID-IMAG in Grenoble (France), Alexandros Stamatakis from the TU München (Germany), and Jarosław Żola from the Iowa State University (USA)

- Minisymposium on Applications of Parallel Computations in Industry and Engineering organized by Raimondas Čiegis from the Vilnius Gediminas Technical University (Lithuania), and Julius Žilinskas from the Institute of Mathematics and Informatics in Vilnius (Lithuania)
- The Second Minisymposium on Interval Analysis organized by Vladik Kreinovich from the University of Texas at El Paso (USA), Paweł Sewastjanow from the Częstochowa University of Technology (Poland), Bartłomiej J. Kubica from the Warsaw University of Technology (Poland), and Jerzy Waśniewski from the Technical University of Denmark (Denmark)
- Workshop on Complex Collective Systems organized by Paweł Topa and Jarosław Wąs from the AGH University of Science and Technology in Cracow (Poland)

Tutorials

The PPAM 2009 meeting began with four tutorials:

- GPUs, OpenCL and Scientific Computing, by Robert Strzodka from the Max-Planck-Institut für Informatik (Germany), Dominik Behr from AMD (USA), and Dominik Göddeke from the University of Dortmund (Germany)
- FPGA Programming for Scientific Computing, by Magnus Peterson from the Synective Labs (Sweden)
- Programming the Cell Broadband Engine, by Maciej Remiszewski from IBM (Poland), and Maciej Cytowski from the University of Warsaw (Poland)
- New Data Structures Are Necessary and Sufficient for Dense Linear Algebra Factorization Algorithms, by Fred Gustavson from the the IBM T.J. Watson Research Center (USA), and Jerzy Waśniewski from the Technical University of Denmark (Denmark)

Best Poster Award

The PPAM Best Poster Award is given to the best poster on display at the PPAM conferences, and was first awarded at PPAM 2009. This award is bestowed by the Program Committee members to the presenting author(s) of the best poster. The selection criteria are based on the scientific content and on the quality of the poster presentation. The PPAM 2009 winner was Tomasz Olas from the Częstochowa University of Technology, who presented the poster "Parallel Adaptive Finite Element Package with Dynamic Load Balancing for 3D Thermomechanical Problems."

New Topics at PPAM 2009

GPU Computing: The recent advances in the hardware, functionality, and programmability of graphics processors (GPUs) have greatly increased their appeal

as add-on co-processors for general-purpose computing. With the involvement of the largest processor manufacturers and the strong interest from researchers of various disciplines, this approach has moved from a research niche to a forward-looking technique for heterogeneous parallel computing. Scientific and industry researchers are constantly finding new applications for GPUs in a wide variety of areas, including image and video processing, molecular dynamics, seismic simulation, computational biology and chemistry, fluid dynamics, weather forecast, computational finance, and many others.

GPU hardware has evolved over many years from graphics pipelines with many heterogeneous fixed-function components over partially programmable architectures towards a more and more homogeneous general purpose design, although some fixed-function hardware has remained because of its efficiency. The general-purpose computing on GPU (GPGPU) revolution started with programmable shaders; later, NVIDIA Compute Unified Device Architecture (CUDA) and to a smaller extent AMD Brook+ brought GPUs into the mainstream of parallel computing. The great advantage of CUDA is that it defines an abstraction which presents the underlying hardware architecture as a sea of hundreds of fine-grained computational units with synchronization primitives on multiple levels. With OpenCL there is now also a vendor-independent high-level parallel programming language and an API that offers the same type of hardware abstraction.

GPU are very versatile accelerators because besides the high hardware parallelism they also feature a high bandwidth connection to dedicated device memory. The latency problem of DRAM is tackled via a sophisticated thread scheduling and switching mechanism on-chip that continues the processing of the next thread as soon as the previous stalls on a data read. These characteristics make GPUs suitable for both compute- and data-intensive parallel processing.

The PPAM 2009 conference recognized the great impact of GPUs by including in its scientific program two major related events: a minisymposium on GPU Computing, and a full day tutorial on "GPUs, OpenCL and Scientific Computing."

The minisymposium received 18 submissions, of which 10 were accepted (55%). The contributions were organized in three sessions. The first group was related to *Numerics*, and comprised the following papers: "Finite Element Numerical Integration on GPUs," " Reduction to Condensed Forms for Symmetric Eigenvalue Problems on Multi-core Architectures," "On Parallelizing the MRRR Algorithm for Data-Parallel Coprocessors," and "A Fast GPU Implementation for Solving Sparse Ill-Posed Linear Equation Systems." The second session dealt with *Applications*. The papers presented were: "Simulations of the Electrical Activity in the Heart with Graphic Processing Units," "Stream Processing on GPUs Using Distributed Multimedia Middleware," and "A GPU Approach to the Simulation of Spatio–temporal Dynamics in Ultrasonic Resonators." Finally, a third session about *General GPU Computing* included presentations of three papers: "Fast In-Place Sorting with CUDA Based on Bitonic Sort," "Parallel Minimax

Tree Searching on GPU," and "Modeling and Optimizing the Power Performance of Large Matrices Multiplication on Multi-core and GPU Platform with CUDA."

The tutorial covered a wide variety of GPU topics and also offered hands-on examples of OpenCL programming that any paticipant could experiment with on their laptop. The morning sessions discussed the basics of GPU architecture, ready-to-use libraries and OpenCL. The afternoon session went in depth on OpenCL and scientific computing on GPUs. All slides are available at http://gpgpu.org/ppam2009.

Complex Collective Systems: Collective aspects of complex systems are attracting an increasing community of researchers working in different fields and dealing with theoretical aspects as well as practical applications. In particular, analyzing local interactions and simple rules makes it possible to model complex phenomena efficiently. Collective systems approaches show great promise in establishing scientific methods that could successfully be applied across a variety of application fields. Many studies in complex collective systems science follow either a cellular automata (CA) method or an agent-based approach. Hybridization between these two complementary approaches gives a promising perspective. The majority of work presented during the workshop on complex collective systems represents the hybrid approach.

We can distinguish four groups of subjects presented during the workshop.

The first group was modeling of pedestrian dynamics: Armin Seyfried from the Juelich Supercomputing Center presented actual challenges in pedestrian dynamics modeling. Another important issue of crowd modeling was also taken into account during the workshop: modeling of stop-and-go waves (Andrea Portz and Armin Seyfried), calibration of pedestrian stream models (Wolfram Klein, Gerta Köster and Andreas Meister), parallel design patterns in a pedestrian simulation (Sarah Clayton), floor fields models based on CA (Ekaterina Kirik, Tat'yana Yurgel'yan and Dmitriy Krouglov), and discrete potential field construction (Konrad Kułakowski and Jarosław Wąs).

The second group dealt with models of car traffic: a fuzzy cellular model of traffic (Bartłomiej Płaczek), and an adaptive time gap car-following model (Antoine Tordeux and Pascal Bouvry).

The third group included work connected with cryptography based on cellular automata: weakness analysis of a key stream generator (Frederic Pinel and Pascal Bouvry), and properties of safe CA-based S-Boxes (Mirosław Szaban and Franciszek Seredyński).

The fourth group dealt with various applications in a field of complex collective systems: frustration and collectivity in spatial networks (Anna Mańka-Krasoń, Krzysztof Kułakowski), lava flow hazard modeling (Maria Vittoria Avolio, Donato D'Ambrosio, Valeria Lupiano, Rocco Rongo and William Spataro), FPGA realization of a CA-based epidemic processor (Pavlos Progias, Emmanouela Vardaki and Georgios Sirakoulis)

Acknowledgements

The organizers are indebted to the PPAM 2009 sponsors, whose support was vital to the success of the conference. The main sponsor was the Intel Corporation. The other sponsors were: Hewlett-Packard Company, Microsoft Corporation, IBM Corporation, Action S.A., and AMD. We thank to all members of the International Program Committee and additional reviewers for their diligent work in refereeing the submitted papers. Finally, we thank all of the local organizers from the Częstochowa University of Technology and Wrocław University of Technology who helped us to run the event very smoothly. We are especially indebted to Grażyna Kołakowska, Urszula Kroczewska, Łukasz Kuczyński, and Marcin Woźniak from the Częstochowa University of Technology; and to Jerzy Świątek, and Jan Kwiatkowski from the Wrocław University of Technology.

PPAM 2011

We hope that this volume will be useful to you. We would like everyone who reads it to feel invited to the next conference, PPAM 2011, which will be held September 11–14, 2011, in Toruń, a city in northern Poland where the great astronomer Nicolaus Copernicus was born.

February 2010

Roman Wyrzykowski
Jack Dongarra
Konrad Karczewski
Jerzy Waśniewski

Organization

Program Committee

Jan Węglarz	Poznań University of Technology, Poland Honorary Chair
Roman Wyrzykowski	Częstochowa University of Technology, Poland Chair
Bolesław Szymański	Rensselaer Polytechnic Institute, USA Vice-Chair
Peter Arbenz	ETH, Zurich, Switzerland
Piotr Bała	N. Copernicus University, Poland
David A. Bader	Georgia Institute of Technology, USA
Michael Bader	TU München, Germany
Mark Baker	University of Reading, UK
Radim Blaheta	Institute of Geonics, Czech Academy of Sciences
Jacek Błażewicz	Poznań University of Technology, Poland
Leszek Borzemski	Wrocław University of Technology, Poland
Pascal Bouvry	University of Luxembourg
Tadeusz Burczyński	Silesia University of Technology, Poland
Jerzy Brzeziński	Poznań University of Technology, Poland
Marian Bubak	Institute of Computer Science, AGH, Poland
Raimondas Čiegis	Vilnius Gediminas Tech. University, Lithuania
Andrea Clematis	IMATI-CNR, Italy
Zbigniew Czech	Silesia University of Technology, Poland
Jack Dongarra	University of Tennessee and ORNL, USA
Maciej Drozdowski	Poznań University of Technology, Poland
Erik Elmroth	Umea University, Sweden
Anne C. Elster	NTNU, Trondheim, Norway
Mariusz Flasiński	Jagiellonian University, Poland
Maria Ganzha	IBS PAN, Warsaw, Poland
Jacek Gondzio	University of Edinburgh, Scotland, UK
Andrzej Gościński	Deakin University, Australia
Laura Grigori	INRIA, France
Frederic Guinand	Université du Havre, France
José R. Herrero	Universitat Politecnica de Catalunya, Barcelona, Spain
Ladislav Hluchy	Slovak Academy of Sciences, Bratislava
Ondrej Jakl	Institute of Geonics, Czech Academy of Sciences
Emmanuel Jeannot	INRIA, France
Grzegorz Kamieniarz	A. Mickiewicz University, Poznań, Poland
Alexey Kalinov	Cadence Design System, Russia
Ayse Kiper	Middle East Technical University, Turkey

Jacek Kitowski	Institute of Computer Science, AGH, Poland
Jozef Korbicz	University of Zielona Góra, Poland
Stanislaw Kozielski	Silesia University of Technology, Poland
Dieter Kranzlmueller	Ludwig Maximillian University, Munich, and Leibniz Supercomputing Centre, Germany
Henryk Krawczyk	Gdańsk University of Technology, Poland
Piotr Krzyżanowski	University of Warsaw, Poland
Jan Kwiatkowski	Wrocław University of Technology, Poland
Giulliano Laccetti	University of Naples, Italy
Marco Lapegna	University of Naples, Italy
Alexey Lastovetsky	University College Dublin, Ireland
Vyacheslav I. Maksimov	Ural Branch, Russian Academy of Sciences
Victor E. Malyshkin	Siberian Branch, Russian Academy of Sciences
Tomas Margalef	Universitat Autonoma de Barcelona, Spain
Ami Marowka	Shenkar College of Engineering and Design, Israel
Norbert Meyer	PSNC, Poznań, Poland
Jarek Nabrzyski	University of Notre Dame, USA
Marcin Paprzycki	IBS PAN and SWPS, Warsaw, Poland
Dana Petcu	Western University of Timisoara, Romania
Enrique S. Quintana-Ortí	Universitat Jaime I, Spain
Yves Robert	Ecole Normale Superieure de Lyon, France
Jacek Rokicki	Warsaw University of Technology, Poland
Leszek Rutkowski	Częstochowa University of Technology, Poland
Franciszek Seredyński	Polish Academy of Sciences and Polish-Japanese Institute of Information Technology, Warsaw, Poland
Robert Schaefer	Institute of Computer Science, AGH, Poland
Jurij Silc	Jozef Stefan Institute, Slovenia
Peter M.A. Sloot	University of Amsterdam, The Netherlands
Masha Sosonkina	Ames Laboratory and Iowa State University, USA
Leonel Sousa	Technical University Lisbon, Portugal
Maciej Stroiński	PSNC, Poznań, Poland
Domenico Talia	University of Calabria, Italy
Andrei Tchernykh	CICESE, Ensenada, Mexico
Carsten Trinitis	TU München, Germany
Roman Trobec	Jozef Stefan Institute, Slovenia
Denis Trystram	ID-IMAG, Grenoble, France
Marek Tudruj	Polish Academy of Sciences and Polish-Japanese Institute of Information Technology, Warsaw, Poland
Pavel Tvrdik	Czech Technical University, Prague
Jens Volkert	Johannes Kepler University, Linz, Austria
Jerzy Waśniewski	Technical University of Denmark
Bogdan Wiszniewski	Gdańsk University of Technology, Poland
Ramin Yahyapour	University of Dortmund, Germany
Jianping Zhu	University of Texas at Arlington, USA

Table of Contents – Part II

Workshop on Scheduling for Parallel Computing (SPC 2009)

The Third Workshop on Language-Based Parallel Programming Models (WLPP 2009)

The Second Workshop on Performance Evaluation of Parallel Applications on Large-Scale Systems

The Fourth Grid Applications and Middleware Workshop (GAMW 2009)

The Fourth Workshop on Large Scale Computations on Grids (LaSCoG 2009)

Workshop on Parallel Computational Biology (PBC 2009)

Minisymposium on Applications of Parallel Computation in Industry and Engineering

Minisymposium on Interval Analysis

Workshop on Complex Collective Systems

Table of Contents – Part I

Parallel/Distributed Architectures and Mobile Computing

Numerical Algorithms and Parallel Numerics

Paralel and Distributed Non-numerical Algorithms

Tools and Environments for Parallel/Distributed/Grid Computing

Applications of Parallel/Distributed Computing

Applied Mathematics and Neural Networks

Minisymposium on GPU Computing

The Second Minisymposium on Cell/B.E. Technologies

Workshop on Memory Issues on Multi- and Manycore Platforms

Fully Polynomial Time Approximation Schemes for Scheduling Divisible Loads

Joanna Berlińska

Faculty of Mathematics and Computer Science,
Adam Mickiewicz University,
Umultowska 87, 61-614 Poznań, Poland
joanna.berlinska@amu.edu.pl

Abstract. In this paper we study divisible loads scheduling in heterogeneous systems with high bandwidth. Divisible loads represent computations which can be arbitrarily divided into parts and performed independently in parallel. We propose fully polynomial time approximation schemes for two optimization problems. The first problem consists in finding the maximum load which can be processed in a given time. It turns out that this problem can be reduced to minimization of a half-product. The second problem is computing the minimum time required to process load of a given size. The FPTAS solving this problem uses a dual approximation algorithm approach.

Keywords: scheduling, divisible loads, FPTAS.

1 Introduction

The divisible load model originated in the late 1980s [1,7] as a tool for modeling processing of big volumes of data. It represents parallel computations which can be divided into pieces of arbitrary sizes. In other words, the grains of parallelism are negligibly small. It is assumed that there are no precedence constraints in the computations, so that the pieces of data can be processed independently in parallel. Divisible load model can be applied to problems such as: search for patterns in text or database files [9], processing measurement data [7,17], image and video processing [12,13,15]. Surveys of divisible load theory and its applications can be found e.g. in [5,14].

The problem of scheduling a divisible application consists in selecting the processors taking part in the computations, choosing the sequence of communications with the processors and the sizes of sent pieces of data. The aim is to process load of data of a given size in the shortest possible time. Alternatively, a dual goal may be to process the largest possible load in a given period of time.

Scheduling divisible loads with no communication startup times was studied in [4,6,3]. In each of these publications it was proved that if there are no communication startup costs, then the load should be sent to the processors in the order of nonincreasing bandwidth. However, the computational complexity of the

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 1–10, 2010.

general problem remained open. In [10] it was proved that scheduling divisible loads in a system with limited memory buffers and with non-zero communication startup times is **NP**-hard. More complexity results, e.g. for systems with release times and with processor deadlines, were shown in [8]. Finally, the **NP**-hardness of divisible loads scheduling was proved in [16].

In this paper we study scheduling divisible loads in heterogeneous systems with communication startup costs and without memory limits. We assume that all working processors have infinite bandwidth, i.e. sending any amount of data to a given processor requires constant time. As [16] proved that our scheduling problem is **NP**-hard even in systems with infinite bandwidth, we propose fully polynomial time approximation schemes for solving the analyzed problems.

The rest of this paper is organized as follows. In Section 2 the analyzed problems are formulated. Sections 3 and 4 describe the algorithms designed to solve the problems. The last section is dedicated to conclusions.

2 Problem Formulation

In this paper we assume that each processor comprises a CPU, memory and a hardware network interface. The CPU and network interface can work in parallel so that simultaneous communication and computation is possible. We assume star interconnection. A set of working processors $\{P_1, \ldots, P_m\}$ is connected to a central server P_0 called originator. Initially, some amount W of load to be processed is located on the originator. We assume that the originator performs no computations. In the opposite case, the computing capability of the originator can be represented as an additional processor. In a general model each working processor P_i, $1 \leq i \leq m$, is described by its computing rate A_i, communication rate C_i and communication startup time S_i. In this paper we assume that each processor has infinite bandwidth, i.e. $C_i = 0$ for $1 \leq i \leq m$. Thus, the time needed to transfer α units of load from P_0 to P_i is S_i, and the time required to process this load on worker P_i is αA_i. We assume that A_i and S_i are nonnegative integers, while W and schedule length are nonnegative rational numbers. We will use the notation $S_{max} = \max_{1 \leq i \leq m} S_i$, $S_{min} = \min_{1 \leq i \leq m} S_i$, $A_{max} = \max_{1 \leq i \leq m} A_i$ and $A_{min} = \min_{1 \leq i \leq m} A_i$.

Let us note that as the bandwidth is infinite and there are no memory limitations, sending more than one piece of data to a processor will not cause starting any computations earlier. Therefore, without loss of generality, we can analyze single-round schedules only.

We study two optimization problems, which can be formulated in the following way (we use the terminology of [16]).

Problem 1. (DLS$\{C_i = 0\}$-Opt$W$)
Given a rational time $T > 0$, m workers, their parameters A_i and S_i for $1 \leq i \leq m$, find the greatest rational number $W_{OPT}(T)$, such that it is possible to process load of size $W_{OPT}(T)$ within time T.

Problem 2. (DLS$\{C_i = 0\}$-Opt$T$)
Given a rational workload size $W > 0$, m workers, their parameters A_i and S_i for $1 \leq i \leq m$, find the smallest rational number $T_{OPT}(W) \geq 0$, such that it is possible to compute the whole load within time $T_{OPT}(W)$.

Let us note that if we analyze the problem DLS$\{C_i = 0\}$-Opt$W$ and $S_i > T$ for some processor P_i, then this processor can process no load in time T. Thus we assume that $S_i \leq T$ for $1 \leq i \leq m$ in the instances of DLS$\{C_i = 0\}$-Opt$W$.

Moreover, if $A_j = 0$ for some processor P_j, then any amount of load can be processed using only processor P_j in time S_j. If we analyze the problem DLS$\{C_i = 0\}$-Opt$W$, we can process infinite load during time T in this case, as we assumed that $S_i \leq T$ for all i. Alternatively, if the studied problem is DLS$\{C_i = 0\}$-Opt$T$, the optimum time needed to process load of size W using processor P_j is S_j. Let T' be the minimum time needed to process load of size W using only processors from the set $P' = \{P_i : \ S_i < S_j\}$. Then the minimum time needed to process W units of load using processors from the whole set $\{P_1, \ldots, P_m\}$ can be computed as $T_{OPT}(W) = \min(S_j, T')$. Therefore, without loss of generality we also assume that $A_i > 0$ for $1 \leq i \leq m$ in the instances of both problems formulated above.

Both problems DLS$\{C_i = 0\}$-Opt$W$ and DLS$\{C_i = 0\}$-Opt$T$ were formulated and studied in [16]. They were shown to be **NP**-hard and pseudo-polynomial dynamic programming algorithms solving them were proposed. Since pseudopolynomial algorithms are in fact exponential, it can be more useful to create fully polynomial time approximation schemes for these problems.

3 FPTAS for the Problem DLS$\{C_i = 0\}$-Opt$W$

The key observation needed to construct an FPTAS solving the problem DLS$\{C_i = 0\}$-Opt$W$ is the following proposition proved in [16].

Proposition 1. *For a given time limit T and set $P' \subseteq \{P_1, \ldots, P_m\}$ of workers taking part in the computation the maximum load is processed if workers are ordered according to nondecreasing values of $S_i A_i$ for $P_i \in P'$.*

Proposition 1 can be proved by the interchange argument: ordering the processors in P' according to nondecreasing $S_i A_i$ does not reduce the amount of processed load.

It follows from Proposition 1 that it is enough to choose the optimal subset of processors taking part in computations to calculate the maximum load which can be processed in a given time T. Namely, let us sort the processors in the order of nondecreasing $S_i A_i$. Let us define a binary vector $\boldsymbol{x} = (x_1, \ldots, x_m)$ in the following way: $x_i = 1$ if processor P_i receives some load to process and $x_i = 0$ in the opposite case. The maximum amount of load that can be processed in time T using processors indicated by the vector $\boldsymbol{x}$ is equal to

$$W_{OPT}(T, \boldsymbol{x}) = \sum_{i=1}^{m} \frac{T x_i}{A_i} - \sum_{i=1}^{m} \sum_{j=i}^{m} \frac{x_i x_j S_i}{A_j}. \tag{1}$$

The first sum in (1) is equal to the load which could be processed if there were no communication delays and the second sum corresponds to the load which is lost because of communication delays (cf. [8,16]).

From now on, we will always assume that processors $P_1, \ldots, P_m$ are sorted according to nondecreasing $S_i A_i$. Our aim is to maximize the size W of load processed in a given period of time T as a function of a binary vector $\boldsymbol{x} = (x_1, \ldots, x_m)$. Instead of maximizing $W(\boldsymbol{x})$, we can minimize the value of $-W(\boldsymbol{x})$. Using the fact that x_i are binary and $x_i^2 = x_i$, we can write

$$-W(\boldsymbol{x}) = -\sum_{i=1}^{m} \frac{T - S_i}{A_i} x_i + \sum_{1 \le i < j \le m} S_i \frac{1}{A_j} x_i x_j. \tag{2}$$

A half-product [2] is a function of the form

$$f(\boldsymbol{x}) = f(x_1, \ldots, x_m) = -\sum_{i=1}^{m} p_i x_i + \sum_{1 \le i < j \le m} q_i r_j x_i x_j. \tag{3}$$

Hence, $-W(\boldsymbol{x})$ is a half-product, with $p_i = \frac{T - S_i}{A_i}$, $q_i = S_i$, $r_j = \frac{1}{A_j}$.

Badics and Boros proposed an FPTAS for minimizing half-products in [2]. They assumed that parameters p_i, q_i, r_i are nonnegative integers for $1 \le i \le m$. In our case all parameters are nonnegative, but $p_i = \frac{T - S_i}{A_i}$ and $r_j = \frac{1}{A_j}$ are not integer. However, the assumption about integrality of p_i and r_i is used neither for proving the correctness of the Badics and Boros algorithm, nor for estimating its running time. Therefore, we can use the algorithm from [2] to minimize the function $-W(\boldsymbol{x})$. The algorithm receives parameters p_i, q_i, r_i and a positive approximation precision $\varepsilon < 1$. It returns a binary vector $\boldsymbol{x}^{\boldsymbol{\varepsilon}} = (x_1^{\varepsilon}, \ldots, x_m^{\varepsilon})$.

For $1 \le k \le m$, let $g_k(\boldsymbol{x}) = -\sum_{i=1}^{k} p_i x_i + \sum_{1 \le i < j \le k} q_i r_j x_i x_j$ and $Q_k(\boldsymbol{x}) = \sum_{i=1}^{k} q_i x_i$. The algorithm can be formulated as follows (cf. [2]):

MINIMIZE-HALF-PRODUCT$((p_1, \ldots, p_m), (q_1, \ldots, q_m), (r_1, \ldots, r_m), \varepsilon)$
STEP 0: Let $\delta > 0$ be defined by the equation $(1 + \delta)^m = 1 + \varepsilon$,
let $Q = \sum_{i=1}^{m} q_i$,
let $N = \lceil \frac{2m \log Q}{\varepsilon} \rceil$,
let $k = 0$ and $\mathcal{X}_0 = \{()\}$.
STEP 1: Let $k = k + 1$, $\mathcal{X}_k = \emptyset$, $t = 0$, $s = 0$,
$\mathcal{L} = \{(y_1, \ldots, y_{k-1}, 0), (y_1, \ldots, y_{k-1}, 1) | (y_1, \ldots, y_{k-1}) \in \mathcal{X}_{k-1}\}$
STEP 2: while $s \le N$ **do**
select $\boldsymbol{z} = (z_1, \ldots, z_k) \in \mathcal{L}$ for which
$t \le Q_k(\boldsymbol{z}) < (1 + \delta)^s$
and for which $g_k(\boldsymbol{z})$ is the smallest among all such $\boldsymbol{z}$.
Let $\mathcal{X}_k = \mathcal{X}_k \cup \{\boldsymbol{z}\}$, $t = (1 + \delta)^s$, $s = s + 1$.
STEP 3: if $k < m$ **goto** STEP 1, **else** goto STEP 4.
STEP 4: Select $\boldsymbol{x}^{\boldsymbol{\varepsilon}} \in \mathcal{X}_m$ with the smallest $g_m(\boldsymbol{x}^{\boldsymbol{\varepsilon}})$, **return** $\boldsymbol{x}^{\boldsymbol{\varepsilon}}$.

It was proved in [2] that

$$f(\boldsymbol{x}^{\boldsymbol{\varepsilon}}) \leq f(\boldsymbol{x}^{*}) + \varepsilon |f(\boldsymbol{x}^{*})|, \tag{4}$$

where $\boldsymbol{x}^{*}$ is a vector minimizing f, and the running time of algorithm MINIMIZE-HALF-PRODUCT is $O(m^2 \log(\sum_{i=1}^{m} q_i)/\varepsilon)$ [2].

Now we propose an algorithm for the problem DLS$\{C_i = 0\}$-Opt$W$. The pseudocode of the algorithm is presented below.

FPTAS-OPT-W(T, ε)
for $i = 1$ **to** m **do**
  $p_i = \frac{T - S_i}{A_i}$
  $q_i = S_i$
  $r_i = \frac{1}{A_i}$
$\boldsymbol{x}^{\boldsymbol{\varepsilon}}$=MINIMIZE-HALF-PRODUCT$((p_1, \ldots, p_m), (q_1, \ldots, q_m), (r_1, \ldots, r_m), \varepsilon)$
return $\boldsymbol{x}_{\boldsymbol{FPTAS}}(T, \varepsilon) = \boldsymbol{x}^{\boldsymbol{\varepsilon}}$, $W_{FPTAS}(T, \varepsilon) = \sum_{i=1}^{m} \frac{T x_i^{\varepsilon}}{A_i} - \sum_{i=1}^{m} \sum_{j=i}^{m} \frac{x_i^{\varepsilon} x_j^{\varepsilon} S_i}{A_j}$

Proposition 2. *The algorithm FPTAS-OPT-W is a fully polynomial time approximation scheme for the problem DLS$\{C_i = 0\}$-Opt$W$.*

Proof. As $\boldsymbol{x}_{\boldsymbol{FPTAS}}(T, \varepsilon)$ is the result of MINIMIZE-HALF-PRODUCT algorithm for the function $-W(\boldsymbol{x})$, we obtain from (4)

$$-W_{FPTAS}(T, \varepsilon) \leq -W_{OPT}(T) + \varepsilon W_{OPT}(T). \tag{5}$$

Consequently,

$$W_{FPTAS}(T, \varepsilon) \geq W_{OPT}(T)(1 - \varepsilon). \tag{6}$$

Moreover, the order of the running time of FPTAS-OPT-W is equal to the running time of MINIMIZE-HALF-PRODUCT, and is equal to at most $O(m^2 \log(\sum_{i=1}^{m} S_i)/\varepsilon) = O(m^2(\log m + \log S_{max})/\varepsilon)$. Hence, the algorithm FPTAS-OPT-W is a fully polynomial time approximation scheme for the problem DLS$\{C_i = 0\}$-Opt$W$. □

4 FPTAS for the Problem DLS$\{C_i = 0\}$-Opt$T$

In order to create an FPTAS for the problem DLS$\{C_i = 0\}$-Opt$T$ we will use the approach of a dual approximation algorithm proposed in [11]. In a dual approximation algorithm the goal is to find a superoptimal infeasible solution of some optimization problem. The performance of the algorithm is measured by the degree of infeasibility allowed.

We will create a dual approximation algorithm for the problem DLS$\{C_i = 0\}$-Opt$W$. Such an algorithm should accept a given period of time T and accuracy ε, and deliver a schedule processing load of size at least $W_{OPT}(T)$ in time not longer than $T(1 + \varepsilon)$. Let us assume that $\varepsilon < 1$ and analyze the following algorithm.

DUAL-OPT-W(T, ε)
call FPTAS-OPT-W$(T, \varepsilon/2)$
return $\boldsymbol{x}_{\boldsymbol{DUAL}}(T, \varepsilon) = \boldsymbol{x}_{\boldsymbol{FPTAS}}(T, \varepsilon/2)$, $W_{DUAL}(T, \varepsilon) = (1+\varepsilon) W_{FPTAS}(T, \varepsilon/2)$

In order to prove that the proposed algorithm is a valid dual approximation algorithm for the problem DLS$\{C_i = 0\}$-Opt$W$, we will use the following fact.

Proposition 3. *If it is possible to process load of size W in time T using the subset of processors indicated by a binary vector $\boldsymbol{x} = (x_1, \ldots, x_m)$, then it is also possible to process load of size $W(1+\varepsilon)$ in time at most $T(1+\varepsilon)$, using the same subset of processors.*

Proof. Let us denote by W' the maximum size of load which can be processed in time $T(1+\varepsilon)$ using the processors indicated by the vector $\boldsymbol{x}$. From (1) we obtain

$$W' = \sum_{i=1}^{m} \frac{T(1+\varepsilon)x_i}{A_i} - \sum_{i=1}^{m}\sum_{j=i}^{m} \frac{x_i x_j S_i}{A_j} \tag{7}$$

and

$$W = \sum_{i=1}^{m} \frac{T x_i}{A_i} - \sum_{i=1}^{m}\sum_{j=i}^{m} \frac{x_i x_j S_i}{A_j}. \tag{8}$$

Therefore,

$$W' = (1+\varepsilon)W + \varepsilon \sum_{i=1}^{m}\sum_{j=i}^{m} \frac{x_i x_j S_i}{A_j} \geq W(1+\varepsilon). \tag{9}$$

□

Note that if $T = T_{OPT}(W)$, then by Proposition 3 we have that load of size $W(1+\varepsilon)$ can be processed in time not longer than $T_{OPT}(W)(1+\varepsilon)$. Hence, we can formulate a corollary:

Corollary 1. *For any numbers $W \geq 0$ and $\varepsilon > 0$ we have*

$$T_{OPT}(W(1+\varepsilon)) \leq T_{OPT}(W)(1+\varepsilon).$$

We will say that an algorithm is a fully polynomial time dual approximation algorithm for a given problem if it is a dual approximation algorithm for this problem with approximation precision ε and its running time is polynomial in both the problem size and $1/\varepsilon$.

Proposition 4. *The algorithm DUAL-OPT-W is a fully polynomial time dual approximation algorithm for the problem DLS$\{C_i = 0\}$-Opt$W$.*

Proof. As $W_{DUAL}(T,\varepsilon) = (1+\varepsilon)W_{FPTAS}(T,\varepsilon/2)$, we get from (6)

$$W_{DUAL}(T,\varepsilon) \geq (1+\varepsilon)W_{OPT}(T)(1-\varepsilon/2) \geq W_{OPT}(T), \tag{10}$$

because $\varepsilon < 1$. Thus, the obtained solution is superoptimal. The time needed to process the load $W_{DUAL}(T,\varepsilon)$ is at most $T(1+\varepsilon)$ by Proposition 3, as it is possible to process load of size $W_{FPTAS}(T,\varepsilon/2)$ in time T.

The running time of the algorithm DUAL-OPT-W is determined by the call of FPTAS-OPT-W$(T,\varepsilon/2)$, whence it is equal to at most $O(m^2(\log m + \log S_{max})/\varepsilon)$. □

We will now construct a fully polynomial time approximation scheme for the problem DLS$\{C_i = 0\}$-Opt$T$ using the above dual approximation algorithm for DLS$\{C_i = 0\}$-Opt$W$ in the binary search process.

FPTAS-OPT-T(W, ε)
$upper = S_{max} + WA_{max}$
$lower = 0$
$LoBo = WA_{min}/m$
while $(upper - lower) > \frac{\varepsilon(1-\varepsilon)}{(2-\varepsilon)} LoBo$
  $T_p = (upper + lower)/2$
  call DUAL-OPT-W(T_p, ε)
  if $W_{DUAL}(T_p, \varepsilon) < W(1 + \varepsilon)$
    then $lower = T_p$
    else $upper = T_p$
call FPTAS-OPT-W$(upper, \varepsilon/2)$
return $\boldsymbol{x} = \boldsymbol{x_{FPTAS}}(upper, \varepsilon/2)$, $T = upper$

Proposition 5. *The algorithm FPTAS-OPT-T is a fully polynomial time approximation scheme for the problem DLS$\{C_i = 0\}$-Opt$T$.*

Proof. Let us start with the observation that at the beginning of the algorithm *upper* and *lower* are trivial upper and lower bounds for $T_{OPT}(W)$. *LoBo* is also a lower bound on $T_{OPT}(W)$ and is positive, since we assumed that $A_i > 0$ for $1 \leq i \leq m$.

First we will analyze the variable *upper* in order to prove that the algorithm always returns a feasible solution. At the beginning of the algorithm $upper = S_{max} + WA_{max}$. If this value is not changed during the binary search, then the algorithm FPTAS-OPT-W is called for parameters $T = upper = S_{max} + WA_{max}$ and approximation precision $\varepsilon/2$ at the end of executing FPTAS-OPT-T algorithm. The obtained schedule will allow for processing load of size at least W, as it is enough to choose any nonempty subset of the workers to process W units of load in time $T = S_{max} + WA_{max}$.

Now let us assume that the value of *upper* is changed at least once to T_p. This happens only if $W_{DUAL}(T_p, \varepsilon) \geq W(1 + \varepsilon)$. Therefore, as

$$W_{DUAL}(T, \varepsilon) = (1 + \varepsilon) W_{FPTAS}(T, \varepsilon/2), \tag{11}$$

we have

$$W_{FPTAS}(upper, \varepsilon/2) = W_{DUAL}(upper, \varepsilon)/(1 + \varepsilon) \geq W \tag{12}$$

at any time during the execution of the algorithm. Hence, the solution obtained by the algorithm FPTAS-OPT-T is always feasible.

Now let us estimate the quality of the obtained solution. We will show that

$$lower \leq T_{OPT}(W)(1 + \frac{\varepsilon}{2 - \varepsilon}) \tag{13}$$

throughout the execution of the program. Since initially $lower = 0$, this condition is true before beginning the binary search. The variable $lower$ is updated to T_p only when $W_{DUAL}(T_p, \varepsilon) < W(1+\varepsilon)$. It follows from (11) that

$$(1+\varepsilon)W_{FPTAS}(lower, \varepsilon/2) < W(1+\varepsilon). \tag{14}$$

Furthermore, from (6) we get

$$(1+\varepsilon)W_{OPT}(lower)(1-\varepsilon/2) < W(1+\varepsilon), \tag{15}$$

$$W_{OPT}(lower) < W/(1-\varepsilon/2) \tag{16}$$

and finally

$$W_{OPT}(lower) < W(1+\frac{\varepsilon}{2-\varepsilon}). \tag{17}$$

Thus, it is impossible to process load $W(1+\frac{\varepsilon}{2-\varepsilon})$ in time $lower$. Hence,

$$lower < T_{OPT}(W(1+\frac{\varepsilon}{2-\varepsilon})). \tag{18}$$

By Corollary 1 we have

$$T_{OPT}(W(1+\frac{\varepsilon}{2-\varepsilon})) \leq T_{OPT}(W)(1+\frac{\varepsilon}{2-\varepsilon}), \tag{19}$$

which proves that (13) is true during the execution of the binary search procedure.

The binary search is finished when $upper \leq lower + \frac{\varepsilon(1-\varepsilon)}{(2-\varepsilon)} LoBo$. Since $LoBo \leq T_{OPT}(W)$, by (13) we get

$$upper \leq T_{OPT}(W)(1+\frac{\varepsilon}{2-\varepsilon}) + \frac{\varepsilon(1-\varepsilon)}{(2-\varepsilon)} T_{OPT}(W) \tag{20}$$

and consequently

$$upper \leq T_{OPT}(W)(1+\varepsilon). \tag{21}$$

Thus FPTAS-OPT-T delivers an ε-approximation of the optimal solution of the problem.

The number of iterations in the binary search is at most equal to $O(\log((S_{max} + WA_{max})/(\frac{\varepsilon(1-\varepsilon)}{(2-\varepsilon)} WA_{min}/m))) = = O(\log S_{max} + \log A_{max} + \log(1/\varepsilon) + \log m + \max(\log W, \log(1/W)))$. The execution time of each iteration is $O(m^2(\log m + \log S_{max})/\varepsilon)$ due to calling the algorithm DUAL-OPT-W. Thus the running time of the whole algorithm FPTAS-OPT-T is at most $O((\log S_{max} + \log A_{max} + \log(1/\varepsilon) + \log m + \max(\log W, \log(1/W)))m^2(\log m + \log S_{max})/\varepsilon)$. □

5 Conclusions

In this paper we studied scheduling divisible loads in a heterogeneous star system with high bandwidth. We formulated two optimization scheduling problems, DLS$\{C_i = 0\}$-Opt$W$ and DLS$\{C_i = 0\}$-Opt$T$. The first problem was computing

the maximum load that can be processed in a given period of time, and the second problem was calculating the minimum time needed to process load of a given size. We proposed fully polynomial time approximation schemes for both of these problems and a dual approximation algorithm for DLS$\{C_i = 0\}$-Opt$W$. Future research may include constructing fully polynomial time approximation schemes for scheduling divisible loads in systems with finite bandwidth characterizing the working processors.

References

1. Agrawal, R., Jagadish, H.V.: Partitioning Techniques for Large-Grained Parallelism. IEEE Transactions on Computers 37(12), 1627–1634 (1988)
2. Badics, T., Boros, E.: Minimization of Half-products. Mathematics of Operations Research 23(3), 649–660 (1988)
3. Beaumont, O., Casanova, H., Legrand, A., Robert, Y., Yang, Y.: Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems. IEEE Transactions on Parallel and Distributed Systems 16(3), 207–218 (2005)
4. Bharadwaj, V., Ghose, D., Mani, V.: Optimal Sequencing and Arrangement in Single-Level Tree Networks with Communication Delays. IEEE Transactions on Parallel and Distributed Systems 5(9), 968–976 (1994)
5. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.G.: Scheduling Divisible Loads in Parallel and Distributed Systems. IEEE Computer Society Press, Los Alamitos (1996)
6. Błażewicz, J., Drozdowski, M.: Distributed Processing of Divisible Jobs with Communication Startup Costs. Discrete Applied Mathematics 76, 21–41 (1997)
7. Cheng, Y.-C., Robertazzi, T.G.: Distributed computation with communication delay. IEEE Transactions on Aerospace and Electronic Systems 24, 700–712 (1988)
8. Drozdowski, M., Lawenda, M.: The combinatorics in divisible load scheduling. Foundations of Computing and Decision Sciences 30(4), 297–308 (2005), http://www.cs.put.poznan.pl/mdrozdowski/txt/divBB2.pdf
9. Drozdowski, M., Wolniewicz, P.: Experiments with scheduling divisible tasks in clusters of workstations. In: Bode, A., Ludwig, T., Karl, W.C., Wismüller, R. (eds.) Euro-Par 2000. LNCS, vol. 1900, pp. 311–319. Springer, Heidelberg (2000)
10. Drozdowski, M., Wolniewicz, P.: Optimum divisible load scheduling on heterogeneous stars with limited memory. European Journal of Operational Research 172, 545–559 (2006)
11. Hochbaum, D., Shmoys, D.: Using dual approximation algorithms for scheduling problems: theoretical and practical results. Journal of the ACM 34(1), 144–162 (1987)
12. Li, X., Bharadwaj, V., Ko, C.C.: Distributed image processing on a network of workstations. International Journal of Computers and Applications 25(2), 1–10 (2003)
13. Lim, T., Robertazzi, T.G.: Efficient parallel video processing through concurrent communication on a multi-port star network. In: Proceedings of the 40th Conference on Information Sciences and Systems, Princeton, NJ, pp. 458–463 (2006)
14. Robertazzi, T.G.: Ten reasons to use divisible load theory. IEEE Computer 36, 63–68 (2003)

15. van der Raadt, K., Yang, Y., Casanova, H.: Practical divisible load scheduling on grid platforms with APST-DV. In: Proceedings of the 19th IPDPS 2005, p. 29b (2005)
16. Yang, Y., Casanova, H., Drozdowski, M., Lawenda, M., Legrand, A.: On the complexity of Multi-Round Divisible Load Scheduling. INRIA Rhône-Alpes, Research Report 6096 (2007)
17. Yu, D., Robertazzi, T.G.: Divisible load scheduling for grid computing. In: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems, PDCS 2003 (2003)

Semi-online Preemptive Scheduling: Study of Special Cases

Tomáš Ebenlendr

Institute of Mathematics, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic
ebik@math.cas.cz

Abstract. We use the duality of linear programing to obtain exact formulas of competitive ratio for the semi-online preemptive scheduling on three and four machines. We use the linear programs from [3]. Namely we consider the online scheduling and the semi-online scheduling with known sum of processing times. We solve the linear programs symbolically by examining all basic solution candidates. All solutions are obtained by a computer but all are verifiable by hand.

1 Introduction

We study the scheduling on *uniformly related machines*. Every machine has its speed s, i.e., processing a job with processing time p in a machine with speed s takes p/s time. *Preemption* is allowed: each job may be divided into several pieces, these pieces can be assigned to different machines in disjoint time slots. The objective is to minimize the *makespan*, i.e., the length of a schedule. In the *online* problem, jobs arrive one-by-one and we need to assign each incoming job without any knowledge of the jobs that arrive later. When a job arrives, its assignment at all times must be given and we are not allowed to change this assignment later. In other words, the online nature of the problem is given by the ordering of the input sequence and it is not related to possible preemptions and the time in the schedule.

The online algorithms are evaluated by their competitive ratio, that is the worst-case ratio between the makespan of the output schedule of the algorithm and the optimal (offline) makespan. I.e., the r-competitive algorithm produces at most r times longer schedule than the best possible schedule for every input sequence. *Semi-online* problems are derived from the original *online* problem by providing some partial information in advance. In this paper we focus on the knowledge of the total processing time. The competitive ratio of studied problems is given by several linear programs, their dimension depends on number of machines quadratically. We deploy a method how to solve small parametrized linear programs. We obtain exact competitive ratios for small numbers of machines (up to 4) for various semi-online scheduling problems this way.

This method is based on duality of linear programing, which says that it suffices to examine all possible basic solutions. From a geometrical point of view, we take all intersections of dimension zero of hyperplanes defined by the linear

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 11–20, 2010.

conditions, and then we test if such an intersection is a feasible and optimal solution. Note that the feasibility and the optimality of such a solution also depends on the actual values of the parameters, so the result typically splits into several cases. Searching through all such intersections would be tedious work as it requires solving a system of linear equations and then examining the feasibility of the result. Most of this work can be automated nowadays as there is various algebraic software available.

2 Definitions of the Problem and Previous Results

We have m machines with speeds $s_1 \geq s_2 \geq \cdots \geq s_m > 0$. We use a shorthand for the total speed of all machines: $S = s_1 + s_2 + \cdots + s_m$. We use special notation for the ratio $\alpha = \frac{S-s_1}{S} = \frac{s_2+s_3+\cdots+s_m}{s_1+s_2+s_3+\cdots+s_m}$ also, as it occurs in resulting formulas. The *input sequence* contains n jobs with processing times $p_1, p_2, \ldots, p_n$. Note that n is unknown to the online algorithm. Again, we use a shorthand $P = p_1 + p_2 + \cdots + p_n$ for the total processing time. The optimal makespan can be computed simply as a maximum of m numbers [5]:

$$C^*_{\max} = \max\left\{p_1^{\max}/s_1, \ldots, (p_1^{\max} + \cdots + p_{m-1}^{\max})/(s_1 + \cdots + s_{m-1}), P/S\right\}\,, \quad (1)$$

where $p_j^{\max}$ is j-th maximal job in the input sequence.

We view every semi-online problem as a *restriction* of set of possible input sequences of the online problem. Then we define a general semi-online input restriction to be simply a set Ψ of allowed input sequences. We call an input sequence a *partial input* if it is a prefix of some input sequence; the set of all partial inputs is denoted $\mathrm{pref}(\Psi)$. Thus the partial inputs are exactly the sequences that the algorithm can see at some point. We restrict the definition of $C^*_{\max}$ only to Ψ-valid sequences of jobs as in [3], i.e., $C^{*,\Psi}_{\max}[\mathcal{J}] = C^*_{\max}[\mathcal{J}]$ for $\mathcal{J} \in \Psi$. Then we extend it to the partial inputs in a natural way, i.e., denoting the least achievable makespan over all valid continuations of such a partial input:

$$C^{*,\Psi}_{\max}[\mathcal{I}] = \inf\{C^*_{\max}[\mathcal{J}] \mid \mathcal{J} \in \Psi \;\&\; \mathcal{I} \text{ is a prefix of } \mathcal{J}\}\,. \quad (2)$$

This simplifies to the similar formula as in (1) for our restrictions. That is crucial, because then we are able to define $C^{*,\Psi}_{\max}$ as the minimum value satisfying several linear inequalities. (We consider the speeds as fixed parameters, because they are given at the start of the algorithm, while the processing times are the variables as the algorithm does not know them in advance.)

We measure how bad are (semi-)online algorithms compared to optimal solution by the competitive ratio. We say that an algorithm is r-competitive if it manages to generate valid schedule with at most r times greater makespan than the makespan of the optimal (offline) schedule. For randomized algorithms we use expectation of the makespan over the random bits of the algorithm for each input sequence.

The exact analysis of scheduling on two machines was given in [4,1,6] for various semi-online problems, and in many more cases for non-preemptive

scheduling. The paper [3] provides the framework to construct the algorithm with best possible competitive ratio for arbitrary number of machines and gives linear programs computing the optimal competitive ratio for several semi-online problems. We solve these linear programs for the cases of up to four machines. Below we list the restrictions studied in our paper.

Online scheduling. Here Ψ contains all sequences. In [2] is designed an optimal online algorithm for all speed vectors, but the competitive ratio is given implicitly by the linear program, which is solved there up to three machines. Here we analyze the competitive ratio of four machines.

Known sum of processing times, $\sum p_j = \bar{P}$. For a given value $\bar{P}$, Ψ contains all sequences with $P = \bar{P}$. The algorithm from [2] is extended in [3] to all semi-online problems studied in this paper. There is also noted that the overall ratio is surprisingly the same as in the general online case, but for $m = 2$, 1-approximation exists. We analyze the cases of $m = 3, 4$.

Other restrictions will be studied in full version of the paper.

The lower bound, as well as matching algorithm can be found in [3]. We consider only nice restrictions, and for these is there proved that the best possible competitive ratio (even for randomized algorithms) can be computed as $r^{\Psi}(\mathbf{s}) = \sup_{\mathcal{J}} \bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$, where:

$$\bar{r}^{\Psi}(\mathbf{s}, \mathcal{J}) = \sum_{j=1}^{n} p_j / \sum_{j=1}^{n} s_{n+1-j} \cdot C^{*,\Psi}_{\max}[\mathcal{J}_{[j]}]\,, \tag{3}$$

where $s_{m+1} = s_{m+2} = \cdots = 0$, $\mathcal{J}$ is a prefix of Ψ-valid input sequence, $\mathcal{J}_{[j]}$ is a sequence containing first j jobs of the input sequence $\mathcal{J}$ and $n = |\mathcal{J}|$. The lower bound uses the argument of the total processing time available to (semi-)online algorithm processing n jobs: after the time $rC^{*,\Psi}_{\max}[J_{[j]}]$ only $n - j$ machines can be used as the algorithm is r-competitive semi-online, thus it has all jobs from $J_{[j]}$ finished, and there are only $n - j$ jobs in $J \setminus J_{[j]}$.

Following simplifications hold for the restrictions studied in this paper: It suffices to consider only $\mathcal{J}$ where jobs are sorted from the smallest to the largest. Also if $n > m$, it suffices to consider sequences with first $n - m$ jobs tiny where only their total processing time is interesting. Then it is easy to construct linear conditions exactly bounding $C^{*,\Psi}_{\max}[J_{[n-m+1]}], \ldots, C^{*,\Psi}_{\max}[J_{[n]}]$, and construct a linear program with the objective function $\bar{r}^{\Psi}(\mathbf{s}, \mathcal{J})$, where the job sizes and the optimal makespans are variables. These programs are already constructed in [3].

3 Online Scheduling

The linear program in variables $q_1, \ldots, q_m$ and $O_1, \ldots, O_m$, follows, each inequality labeled with corresponding dual variable ($z_?$). The value of the optimal solution is the competitive ratio of the problem for m machines. This program is already solved for $m \leq 3$ in [2].

$$\begin{array}{lll}
\textbf{maximize} & r = q_1 + \cdots + q_m & \\
\textbf{subject to} & 1 = s_1 O_m + s_2 O_{m-1} + \cdots + s_m O_1 & (z_{norm}) \\
& q_1 + \cdots + q_k \le (s_1 + \cdots + s_m) O_k & (z_k) \quad 1 \le k \le m \\
& q_j + \cdots + q_k \le (s_1 + \cdots + s_{k-j+1}) O_k & (z_{j,k}) \quad 2 \le j \le k \le m \\
& q_j \le q_{j+1} & (z_{\le,j}) \quad 2 \le j \le m-1 \\
& 0 \le q_1\,, \quad 0 \le q_2 & (z_{0,1})\,, \ (z_{0,2})
\end{array} \tag{4}$$

So we have the linear program and we solve it for all speed combinations of four machines ($m = 4$).

The list of the cases follows. We list not only the resulting competitive ratio and the validity domain of the case, (i.e., the conditions that define this domain), but also the sizes of the jobs in the input sequence proving the lower bound and also the dual coefficients proving that no better bound can be obtained from (3). Note that the algorithm from [2] matches exactly this lower bound, i.e., if it fails to achieve some competitive ratio, then it is due to the bound (3). (Recall the definition $\alpha = \frac{S-s_1}{S}$.)

Case I Ratio: $r = S/D\,, \quad D = s_1 + \alpha s_2 + \alpha^2 s_3 + \alpha^3 s_4$

Conditions:	Jobs:	U.b. coefficients:
(A+: II) $s_2 \ge \alpha s_1$	$q_1 = \alpha^2(S-s_1)/D$	$z_1 = z_{2,2} = s_4/D$
(B+: IV) $s_2+s_3 \ge (\alpha+\alpha^2)s_1$	$q_2 = \alpha^2 s_1/D$	$z_2 = (s_3-(1-\alpha)s_4)/D$
Nonbasic dual vars:	$q_3 = \alpha s_1/D$	$z_3 = (s_2-(1-\alpha)(s_3+\alpha s_4))/D$
$z_{0,1}, z_{0,2}, z_{\le,2}, z_{\le,3}$,	$q_4 = s_1/D$	$z_4 = (s_1-(1-\alpha)(s_2+\alpha s_3+\alpha^2 s_4))/D$
$z_{2,3}, z_{2,4}, z_{3,4}$		$z_{3,3} = (s_3+\alpha s_4)/D$
		$z_{4,4} = (s_2+\alpha s_3+\alpha^2 s_4)/D$

We use a shorthand D for the common denominator of all formulas in our case description. *Conditions* state for which values of parameters is this case optimal. The label (A+: II) of a condition should be read: The Case I is adjacent to the Case II, where the opposite condition (A-): $s_2 \le \alpha s_1$ holds. *Jobs* give the main input sequence for lower bound. Note that the adversary may stop the sequence after any number of jobs. (In the general semi-online case the adversary may need to submit some more jobs, so that the input will be in Ψ, while maintaining $C^{*,\Psi}_{\max}$. But this is not needed in online scheduling.) The *nonbasic dual variables* are labels of inequalities, which we allow to be not tight, i.e., all other inequalities are turned into equations. The *upper bound coefficients* are values of nonzero dual variables. These give the matching upper bound on the competitive ratio, which is obtained by summing up the corresponding inequalities multiplied by these coefficients. Note that all nonbasic dual variables have coefficients of zero value (and thus are not listed), because of the dual complementary slackness of linear programing.

Now we show how to check the correctness of Case I. The correctness of all other cases and all other restrictions in this paper can be checked in the same way. First we check the value of the objective function (using $s_1 = (1-\alpha)S$):

$$q_1 + q_2 + q_3 + q_4 \;=\; \alpha^2 S/D + \alpha(1-\alpha)S/D + (1-\alpha)S/D \;=\; S/D \;=\; r$$

Then we compute the values of the optima variables $O_1, \ldots, O_4$. We know that basic inequalities are satisfied by equality:

$$\begin{aligned}
O_1 &\overset{(z_1)}{=} q_1/S = \alpha^3/D \\
O_2 &\overset{(z_{2,2})}{=} q_2/s_1 = \alpha^2/D = (q_1+q_2)/S \overset{(z_2)}{=} O_2 \\
O_3 &\overset{(z_{3,3})}{=} q_3/s_1 = \alpha/D = (q_1+q_2+q_3)/S \overset{(z_3)}{=} O_3 \\
O_4 &\overset{(z_{4,4})}{=} q_4/s_1 = 1/D = (q_1+q_2+q_3+q_4)/S \overset{(z_4)}{=} O_4 \,.
\end{aligned}$$

The equal signs are labeled by labels of used equalities. Similarly the inequality signs are labeled by labels of sufficient conditions in following text.

We can also easily verify that the equation (z_{norm}) holds. We check the remaining (i.e., nonbasic) inequalities:

$$\begin{aligned}
(z_{2,3}) \quad & q_2+q_3 = (\alpha+\alpha^2)s_1/D \quad \leq^A \quad \alpha(s_1+s_2)/D = (s_1+s_2)O_3 \\
(z_{2,4}) \quad & q_2+q_3+q_4 = (1+\alpha+\alpha^2)s_1/D \leq^B (s_1+s_2+s_3)/D = (s_1+s_2+s_3)O_4 \\
(z_{3,4}) \quad & q_3+q_4 = (1+\alpha)s_1/D \quad \leq^A \quad (s_1+s_2)/D = (s_1+s_2)O_4 \,.
\end{aligned}$$

We get $(z_{0,1}), (z_{0,2}), (z_{\leq,2})$ and $(z_{\leq,3})$ trivially from $\alpha \leq 1$ and $S \geq s_1$. Thus we know that our solution is feasible when A+ and B+ holds, so all algorithms are at least r-competitive for such sets of speeds. The sequence that proves this is for example: $p_1 = \cdots = p_4 = q_1/4, p_5 = q_2, p_6 = q_3, p_7 = q_4$.

Now we check the optimality of our solution. We check that all upper bound coefficients are nonnegative with the exception of z_{norm}:

$$\begin{aligned}
z_2 D &= (s_3 - s_4) + \alpha s_4 \;\geq\; 0 \\
z_3 D &= (s_2 - s_3) + \alpha(s_3 - s_4) + \alpha^2 s_4 \;\geq\; 0 \\
z_4 DS &= s_1 S - s_1(s_2 + \alpha s_3 + \alpha^2 s_4) \;\geq\; 0 \,.
\end{aligned}$$

The coefficient z_{norm} is allowed to be negative, as (z_{norm}) is an equation. So we have all inequality coefficients nonnegative, thus we add up the inequalities multiplied by their respective coefficients, and we use the resulting inequality:

$$\begin{aligned}
&q_1+q_2+q_3+q_4 = q_1(z_1+z_2+z_3+z_4)+q_2(z_{2,2}+z_2+z_3+z_4)+q_3(z_{3,3}+z_3+z_4)+q_4(z_{4,4}+z_4) \\
&\leq O_1 S z_1 + O_2(S z_2 + s_1 z_{2,2}) + O_3(S z_3 + s_1 z_{3,3}) + O_4(S z_4 + s_1 z_{4,4}) \\
&= (O_1 s_4 + O_2 s_3 + O_3 s_2 + O_4 s_1) S/D + (O_1 s_4 + O_2 s_3 + O_3 s_2 + O_4 s_1 - 1) z_{norm} \\
&= S/D \,.
\end{aligned}$$

This proves the optimality of our solution, i.e., there is no better solution and the algorithm from [2] is r-competitive.

Now we continue the list of the cases:

Case II Ratio: $r = S^2/D$, $D = \sum_{i=1}^{4}\sum_{j=i}^{4} s_i s_j + \alpha(s_3+s_4)s_4 - s_4^2$

Conditions:	Jobs:	U.b. coefficients:
(A-: I) $s_2 \leq \alpha s_1$	$q_1 = (s_3+s_4)(S-s_1)/D$	$z_1 = z_{2,3} = s_4 S/D$
(C+: III) $s_2+s_3 \geq \alpha(s_1+s_2)$	$q_2 = (s_3+s_4)s_1/D$	$z_2 = z_{3,4} = s_3 S/D$
Nonbasic dual vars:	$q_3 = s_2 S/D$	$z_3 = (s_2 S-(s_1+s_2)s_4)/D$
$z_{0,1}, z_{0,2}, z_{\leq,2}, z_{\leq,3},$	$q_4 = s_1 S/D$	$z_4 = (s_1(s_1+s_4)-s_2 s_3$
$=z_{2,2}, z_{2,4}, z_{3,3}$		$-(1-\alpha)(s_3+s_4)s_4)/D$
		$z_{4,4} = (s_2 S+(s_3+s_4)s_4)/D$

Case III Ratio: $r = S^2/D$
$D = \sum_{i=1}^{4}\sum_{j=i}^{4} s_i s_j$

Conditions: (A-: IV) $s_2 \le \alpha s_1$
(C-: II) $s_2+s_3 \le \alpha(s_1+s_2)$

Nonbasic dual vars: $z_{0,1}, z_{0,2}, z_{\le,2}, z_{\le,3}, z_{2,2}, z_{2,3}, z_{3,3}$

Jobs:	U.b. coefficients:
$q_1 = s_4 S/D$	$z_1 = z_{2,4} = s_4 S/D$
$q_2 = s_3 S/D$	$z_2 = z_{3,4} = s_3 S/D$
$q_3 = s_2 S/D$	$z_3 = z_{4,4} = s_2 S/D$
$q_4 = s_1 S/D$	$z_4 = (s_1 S - \sum_{i=2}^{4}\sum_{j=1}^{i-1} s_i s_j)/D$

Case IV Ratio: $r = S/D$
$D = s_1 + \alpha s_2 + \alpha^2 s_3 + s_4^2/S$

Conditions: (A+: III) $s_2 \ge \alpha s_1$
(B-: I) $s_2+s_3 \le (\alpha+\alpha^2)s_1$

Nonbasic dual vars: $z_{0,1}, z_{0,2}, z_{\le,2}, z_{\le,3}, z_{2,2}, z_{2,3}, z_{3,4}$

Jobs:	U.b. coefficients:
$q_1 = s_4/D$	$z_1 = z_{2,4} = s_4/D$
$q_2 = (\alpha(S-s_1)-s_4)/D$	$z_2 = z_{3,3} = s_3/D$
$q_3 = \alpha s_1/D$	$z_3 = (s_2-(1-\alpha)s_3)/D$
$q_4 = s_1/D$	$z_4 = (s_1 - s_4(S-s_4)/S - (1-\alpha)(s_2+s_3\alpha))/D$
	$z_{4,4} = (s_2+\alpha s_3)/D$

We should also check that all listed cases cover whole space of the valid parameters (the speeds of the machines). This is easy, as A splits the space to the cases I+IV and the cases II+III. Then, I and IV are separated by B and fully cover the halfspace A+. Similarly II and III are separated by C and fully cover A-.

4 Known Sum of Processing Times, $\sum p_j = \bar{P}$

Here we are given a value $\bar{P}$ and Ψ contains all $\mathcal{J}$ with $P = \bar{P}$. Here we have to solve $n-1$ linear programs for each $n < m$, and take the maximum of their solutions. The linear program for arbitrary n follows. Note that the shorthand S sums all speeds of the machines, i.e., including $s_{n+1}, \ldots, s_m$.

$$\begin{array}{rrll}
\textbf{maximize} & r = q_1 + q_2 + \cdots + q_n & & \\
\textbf{subject to} \quad 1 &= s_1 O_n + s_2 O_{n-1} + \cdots + s_n O_1 & (z_{norm}) & \\
q_1 + \cdots + q_n &\le S O_k & (z_k) & 1 \le k \le n-1 \\
q_j + \cdots + q_k &\le (s_1 + \cdots + s_{k-j+1}) O_k & (z_{j,k}) & 1 \le j \le k \le n \\
q_k &\le q_{k+1} & (z_{\le,k}) & 1 \le k \le n-1 \\
0 &\le q_1 & (z_0)\,. &
\end{array} \tag{5}$$

We omit the inequality (z_n) as it is implied by $(z_{1,n})$. (The implication follows trivially from $S = s_1 + \cdots + s_n + s_{n+1} + \cdots + s_m$.)

The linear program is trivial for $n = 1$, and we conclude that for $m = 2$ the approximation ratio is equal to 1, i.e., there is an optimal algorithm.

***m* = 3.** For $m = 3$, it remains to solve the linear program for $n = 2$. The ratio splits to two cases:

Case I Ratio: $r = (s_1+s_2)S/D$
$D = s_2(s_1+s_2) + s_1 S$

Conditions: (A+: II) $s_1(s_1+s_2) \ge s_2 S$

Nonbasic dual vars: $z_0, z_{\le,1}, z_{1,1}$

Jobs:	U.b. coefficients:
$q_1 = s_2 S/D$	$z_1 = s_2(s_1+s_2)/D$
$q_2 = s_1 S/D$	$z_{1,2} = s_1 S/D$

Case II Ratio: $r = s_1(s_1+s_2)/D$
$D = s_1^2 + s_2^2$

Conditions: (A-: I) $s_1(s_1+s_2) \le s_2 S$

Nonbasic dual vars: $z_0, z_{\le,1}, z_1$

Jobs:	U.b. coefficients:
$q_1 = s_1 s_2/D$	$z_{1,2} = s_1 S/D$
$q_2 = s_1^2/D$	$z_{1,1} = z_{2,2} = s_2(s_1+s_2)/D$

$m = 4$. Here we solve the linear program for $n = 3$. Note, that the competitive ratio is the maximum of results of linear programs for all $n < m$.

Case I Ratio: $r = (s_1 + s_2 + s_3)S/D$, $D = (s_2 + s_3)(s_1 + s_2 + s_3) + s_1 S$

Conditions:
(A+: II,III) $(s_1+s_2)(s_1+s_2+s_3) \geq (s_2+s_3)S$

Jobs:
$q_1 = q_2 = (s_2+s_3)S/(2D)$
$q_3 = s_1 S/D$

U.b. coefficients:
$z_2 = s_2(s_1+s_2+s_3)/D$
$z_{1,3} = s_1 S/D$
$z_1 = s_3(s_1+s_2+s_3)/D$

Nonbasic dual vars:
$z_0, z_{\leq,2}, z_{1,1}, z_{1,2}, z_{2,2}, z_{2,3}$

Case II Ratio: $r = (s_1 + s_2)(s_1 + s_2 + s_3)S/D$
$D = (s_1^2 + s_2(s_1 + s_2 + s_3))S + s_3(s_1 + s_2)(s_1 + s_2 + s_3)$

Conditions: (implicit: F+)
(A-: I) $(s_1+s_2)(s_1+s_2+s_3) \leq (s_2+s_3)S$
(B+: III) $s_1 s_3 \geq s_2^2$
(C+: IV,V) $s_1(s1+s_2+s_3) \geq s_3 S$

U.b. coefficients:
$z_1 = s_3(s_1+s_2)(s_1+s_2+s_3)/D$
$z_{1,2} = z_{3,3} = s_2(s_1+s_2+s_3)S/D$
$z_{1,3} = s_1^2 S/D$

Jobs:
$q_1 = s_3(s_1+s_2)S/D$
$q_2 = s_2(s_1+s_2)S/D$
$q_3 = s_1(s_1+s_2)S/D$

Nonbasic dual vars: $z_0, z_{\leq,1}, z_{\leq,2}, z_2, z_{1,1}, z_{2,2}$

Case III Ratio: $r = (s_1 + s_2)(s_1 + s_2 + s_3)S/D$
$D = (s_1^2 + s_2(s_1 + s_2 + s_3))S + s_3(s_1 + s_2)(s_1 + s_2 + s_3)$

Conditions:
(A-: I) $(s_1+s_2)(s_1+s_2+s_3) \leq (s_2+s_3)S$
(B-: II) $s_1 s_3 \leq s_2^2$
(D+: VII) $s_1(s_1+s_2)(s1+s_2+s_3) \geq s_2(s_2+s_3)S$

Jobs:
$q_1 = s_2(s_2+s_3)S/D$
$q_2 = s_1(s_2+s_3)S/D$
$q_3 = s_1(s_1+s_2)S/D$

Nonbasic dual vars:
$z_0, z_{\leq,1}, z_{\leq,2}, z_2, z_{1,1}, z_{2,3}$

U.b. coefficients:
$z_1 = s_3(s_1+s_2)(s_1+s_2+s_3)/D$
$z_{1,2} = z_{3,3} = s_2(s_1+s_2+s_3)S/D$
$z_{1,3} = s_1^2 S/D$

Case IV Ratio: $r = (s_1 + s_2)^2 S/D$
$D = (s_1^2 + s_1 s_2 + s_2^2)(S - s_1) + (s_1 + s_2)(s_1 s_2 + s_1 s_3 + s_2 s_3)$

Conditions:
(implicit: A-,B+)
(C-: II)) $s_1(s_1+s_2+s_3) \leq s_3 S$
(E-: V)) $s_1^3 \leq s_3(s_1+s_2)^2$
(F+: VI)) $s_1(s_1^2+s_1 s_2+s_2^2) \geq s_2^2 S$

Jobs:
$q_1 = s_1(s_1+s_2)^2/D$
$q_2 = s_2(s_1+s_2)(S-s_1)/D$
$q_3 = s_1(s_1+s_2)(S-s_1)/D$

Nonbasic dual vars:
$z_0, z_{\leq,1}, z_{\leq,2}, z_2, z_{1,3}, z_{2,2}$

U.b. coefficients:
$z_1 = (s_3(s_1+s_2)^2 - s_1^3)/D$
$z_{1,1} = z_{2,3} = s_1^2 S/D$
$z_{1,2} = z_{3,3} = s_2(s_1+s_2)S/D$

Case V Ratio: $r = s_1(s_1 + s_2)(s_1 + s_2 + s_3)/D$
$D = s_1(s_1 + s_2)(s_1 + s_2 + s_3) - (s_1 - s_3)(s_1^2 - s_2 s_3)$

Conditions:
(implicit: A-)
(B+: VII) $s_1 s_3 \geq s_2^2$
(C-: II) $s_1(s_1+s_2+s_3) \leq s_3 S$
(E+: IV,VI) $s_1^3 \geq s_3(s_1+s_2)^2$

Jobs:
$q_1 = s_3 s_1(s_1+s_2)/D$
$q_2 = s_2 s_1(s_1+s_2)/D$
$q_3 = s_1^2(s_1+s_2)/D$

Nonbasic dual vars:
$z_0, z_{\leq,1}, z_{\leq,2}, z_1, z_2, z_{2,2}$

U.b. coefficients:
$z_{1,1} = z_{2,3} = s_3(s_1+s_2)(s_1+s_2+s_3)/D$
$z_{1,2} = z_{3,3} = s_1 s_2(s_1+s_2+s_3)/D$
$z_{1,3} = (s_1^3 - (s_1+s_2)^2 s_3)/D$

Case VI Ratio: $r = s_1(s_1^2 + s_1s_2 + s_2^2)/D$, $D = s_1^3 + s_2^2(s_1 + s_3)$

Conditions:	(implicit: A-,C-)	Jobs:	U.b. coefficients:
(B+: VII)	$s_1s_3 \geq s_2^2$	$q_1 = s_1s_2^2/D$	$z_{1,1} = s_3(s_1^2+s_1s_2+s_2^2)/D$
(E-: V)	$s_1^3 \leq s_3(s_1+s_2)^2$	$q_2 = s_1^2s_2/D$	$z_{1,2} = s_1(s_1^2+s_2^2-s_3(s_1+s_2))/D$
(F-: VI)	$s_1(s_1^2+s_1s_2+s_2^2) \leq s_2^2S$	$q_3 = s_1^3/D$	$z_{2,2} = (s_3(s_1+s_2)^2-s_1^3)/D$
Nonbasic dual vars:			$z_{2,3} = s_1(s_1^2-s_2s_3)/D$
$z_0, z_{\leq,1}, z_{\leq,2}, z_1, z_2, z_{1,3}$			$z_{3,3} = s_2(s_1s_2+s_1s_3+s_2s_3)/D$

Case VII Ratio: $r = s_1(s_1 + s_2)(s_1 + s_2 + s_3)/D$
$D = s_1^2(s_1 + s_2) + s_2(s_1 + s_3)(s_2 + s_3)$

Conditions:	(B-: V,VI)	$s_1s_3 \leq s_2^2$	Nonbasic dual vars:
(implicit: A-)	(D-: III)	$s_1(s_1+s_2)(s1+s_2+s_3) \leq s_2(s_2+s_3)S$	$z_0, z_{\leq,1}, z_{\leq,2}, z_1, z_2, z_{2,3}$
U.b. coefficients:	$z_{1,1} = z_{2,2} = s_3(s_1+s_2)(s_1+s_2+s_3)/D$		Jobs:
	$z_{1,2} = s_1(s_2-s_3)(s_1+s_2+s_3)/D$		$q_1 = s_1s_2(s_2+s_3)/D$
	$z_{1,3} = s_1(s_1^2-s_2s_3)/D$		$q_2 = s_1^2(s_2+s_3)/D$
	$z_{3,3} = s_1(s_1+s_3)(s_1+s_2+s_3)/D$		$q_3 = s_1^2(s_1+s_2)/D$

5 Techniques for Solving the Parametrized LPs

The solutions above were obtained by search through a large number of possibilities (10^4). This is impossible to do manually and thus we developed a method how to filter out most of the invalid possibilities by a computer. Remaining number of possible solutions is small enough to be solved by a man. Mathematical software Maple 9.5 was used, but any modern algebraical software should do the task. The description of the method follows.

We use the notation $\{\max \mathbf{c}^T\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b}\}$ for the primal linear program, and $\{\min \mathbf{y}^T\mathbf{b} \mid \mathbf{y}^TA = \mathbf{c}^T, \mathbf{y} \geq \mathbf{0}\}$ for the corresponding dual linear program. W.l.o.g., the number of the primal variables (the dimension of $\mathbf{x}$) is smaller than or equal to the number of the dual variables (the dimension of $\mathbf{y}$).

We use the duality of the linear programing [7], i.e., if there is an optimal solution to the primal program, then there is a pair of the primal and the dual basic solutions which are optimal. Then we use the dual complementary slackness: the primal inequalities not satisfied by equality imply zero values of the corresponding dual variables. We also use the fact it suffices to examine the vertices of the polytope. We know that the result is bounded, because there is universal upper bound on competitive ratio and the input sequence with at least one nonzero job gives a positive lower bound. Thus we take the set of the dual variables and we generate all subsets of cardinality equal to the dimension of the linear program (which is the number of the primal variables for all linear programs that we examine). We get all points of intersections of the conditions this way. We call them the solution candidates. Then we have to find the points that are feasible and optimal for some valid values of input parameters. From the duality slackness conditions, there is one to one mapping between the solution candidates of primal program and the solution candidates of the dual one.

Now we stick to one arbitrary fixed subset Y of the dual variables and we describe how the computer helps us to examine the solution pair induced by this

subset. Let Y be a square matrix with $y_{i,i} = 1$ if $i \in Y$ and $y_{i,j} = 0$ otherwise. Now we have the primal candidate solution satisfying system of equations $YA\mathbf{x} = Y\mathbf{b}$ and the candidate dual solution satisfying $\mathbf{y}^T A = \mathbf{c}^T$ & $\mathbf{y}^T(I - Y) = 0$. I.e., we set a primal inequality to equality if it corresponds to some selected dual variable. We set the not selected dual variables to zero and we change the dual inequalities to equations. We solve these two sets of linear equations using Maple, and then we examine this solution pair.

At first we try to filter out the infeasible solution pairs. So how is our domain of feasibility defined? The primal solution is feasible when all inequalities (namely the inequalities corresponding to the not selected dual variables) are satisfied. The dual solution is feasible when all variables are nonnegative.

It may happen that either the primal or the dual candidate solution does not exist, i.e., the system of equations has no solution. But we already know that optimal competitive ratio is bounded, which contradicts feasibility of such a solution pair, we eliminate it. The positive lower bound also contradicts feasibility of the solution pair, which has zero value of the resulting competitive ratio.

Now we examine the domain of feasibility of both primal and dual candidate solutions. We developed a heuristic that uses the inequalities between the parameters (i.e., the speeds of the machines, the inequalities are $s_i \geq s_{i+1}$ and $s_i \geq 0$) and determines the validity of the given inequality. The outcome of this heuristic is one of the three cases: (i) surely always valid, (ii) surely always invalid or (iii) there may be values of parameters for which is the inequality valid and another values for which is the inequality invalid. Note that inequality that is always valid or always invalid may be so complex that our heuristic evaluates it as the third case. Our heuristic also uses the factorization of polynomials to eliminate factors that are always positive or always negative. This decreases the polynomial degree of the inequality. So our heuristic may return a simpler inequality that is equivalent to the original one in the third case.

The feasibility domain is given as a set of inequalities. We use our heuristic on them. If we find an inequality that is always invalid (i.e., for all valid values of parameters), we eliminate such a solution pair for infeasibility. If we do not eliminate the pair, we eliminate the inequalities that are surely always valid, and we replace the inequalities with the simpler versions, if our heuristic finds some. We try to further eliminate the solution pair for infeasibility, or to simplify the inequalities defining the feasible region.

Now we are done with single inequalities. So we consider pairs of inequalities. We already have the set of inequalities reduced only to inequalities that may be invalid for some values of parameters. A pair of such inequalities may be in contradiction, then the solution pair is infeasible. Or one inequality may be implied by another one, then we reduce the set of inequalities defining the feasible region. To test the contradiction or the implication, we simply try to add or subtract the conditions, one of them possibly multiplied by a factor from some small predefined set of factors. We test the result using our heuristic again.

After all these computations are done, there remain several solution pairs that have to be eliminated by hand. There may be a condition too complex for our

heuristic, or there may be three or more conditions in contradiction. Also, our set of factors for testing contradiction may not contain the factor needed to prove the contradiction of the two conditions. Number of these solution pairs vary, but in general there were fewer such solution pairs than the valid ones. The tools that we developed for the automated part are also useful here.

At last, sometimes there are more solution pairs with the same competitive ratio. Domains of feasibility of such pairs may overlap, and sometimes they do. But in all the examined cases there was one or several non overlapping solution pairs, that covered the whole domain of such a formula for the competitive ratio, while the remaining pairs were superfluous. We finish our inspection of solutions by finding which cases are neighbor by which inequality, thus it can be easily verified (even without using the computer), that the feasibility domains cover the set of all valid values of parameters (the speeds of machines).

Conclusions. We solve the special cases of $m = 3$ and $m = 4$ for the online scheduling and for semi-online scheduling with known sum of processing times. The online scheduling on four machines was more demanding on the computer, as there was $\binom{12}{5} = 792$ basic solution candidates, all but six were found infeasible automatically by computer. Four of the remaining six give the four cases of the optimal competitive ratio. On the other hand, the semi-online scheduling with known sum of processing times for prefixes of three jobs (the hardest case when solving four machines), has only $\binom{10}{5} = 252$ basic solution candidates, all but 20 were found infeasible, and the remaining 20 cases had to be processed manually. Only seven of them are relevant for the optimal competitive ratio. Solving these cases exactly is now possible only using our method (or a similar one), because of the amount of mathematical (algebraic) operations that must be done to go through all the cases. Our method can be further improved, but this will not improve our results dramatically, because of the exponential case explosion. This work also shows that the complexity of exact formulas of competitive ratio grows dramatically with the number of machines.

References

1. Du, D.: Optimal preemptive semi-online scheduling on two uniform processors. Inform. Process. Lett. 92(5), 219–223 (2004)
2. Ebenlendr, T., Jawor, W., Sgall, J.: Preemptive online scheduling: Optimal algorithms for all speeds. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 327–339. Springer, Heidelberg (2006)
3. Ebenlendr, T., Sgall, J.: Semi-Online Preemptive Scheduling: Online Algorithm for All Variants. In: Proc. 26st Symp. on Theoretical Aspects of Comput. Sci. (STACS), Dagstuhl Seminar Proceedings, vol. 9001, pp. 346–360, IBFI (2009)
4. Epstein, L., Favrholdt, L.M.: Optimal preemptive semi-online scheduling to minimize makespan on two related machines. Oper. Res. Lett. 30, 269–275 (2002)
5. Horwath, E., Lam, E.C., Sethi, R.: A level algorithm for preemptive scheduling. J. ACM 24, 32–43 (1977)
6. Jiang, Y., He, Y.: Optimal semi-online algorithms for preemptive scheduling problems with inexact partial information. Theoret. Comput. Sci. 44(7-8), 571–590 (2007)
7. Vazirani, V.V.: Approximation algorithms. In: LP duality, ch. 12. Springer, Heidelberg (2001)

Fast Multi-objective Reschulding of Grid Jobs by Heuristics and Evolution

Wilfried Jakob, Alexander Quinte, Karl-Uwe Stucky, and Wolfgang Süß

Karlsruhe Institute of Technology (KIT), Institute for Applied Computer Science,
P.O. Box 3640, 76021 Karlsruhe, Germany
{wilfried.jakob,alexander.quinte,uwe.stucky,wolfgang.suess}@kit.edu

Abstract. Scheduling of jobs to a computational grid is a permanent process due to the dynamic nature of the grid and the frequent arrival of new jobs. Thus, a permanent rescheduling of already planned and new jobs must be performed. This paper will continue and extend previous work, which focused on the tuning of our **G**lobal **O**ptimising **R**esource **B**roker and **A**llocator GORBA in a static planning environment. A formal definition of the scheduling problem and a classification will be given. New heuristics for rescheduling exploiting the "old plan" will be introduced and it will be investigated how they contribute to the overall planning process. Furthermore, the maximal possible load, which can be handled within the given time frame of three minutes, will be examined for a grid of growing size of up to 6000 grid jobs and 600 resources.

1 Introduction

A computational grid can be regarded a virtualised and distributed computing centre [1]. Users describe their *application jobs*, consisting of one or more basic *grid jobs*, by workflows, each of which may be regarded a directed acyclic graph defining precedence rules between the grid jobs. The users state which resources like software, data storage, or computing power are needed to fulfil their grid jobs. Resources may need other ones. A software tool, for instance, may require a certain operating system and appropriate computer hardware to run on. This leads to the concept of co-allocation of resources. Furthermore, users will give due dates, cost budgets and may express a preference for cheap or fast execution [2]. For planning, execution times of the grid jobs are needed. In case of entirely new jobs, this can be done by estimations or by the use of prediction systems only. Otherwise, values coming from experience can be used. The grid middleware is expected to support this by providing runtimes and adding them to the workflow for further usage. According to the policy of their owners, resources are offered at different costs depending on e.g. day time or day of the week and their usage may be restricted to certain times. In addition, heterogeneous resources usually differ in performance as well as cost-performance ratios.

To fulfil the different needs of resource users and providers, the following four objectives are considered: *completion time* and *costs* of each application job measured as fulfilment of user-given limits and averaged, and to meet the

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 21–30, 2010.

demands of resource providers, the *total makespan* of all application jobs and the ratio of *resource utilisation*. Some of these criteria like costs and time are obviously conflicting.

As grid jobs are assumed to require computing time in the magnitude of several minutes at the minimum, a certain but limited time frame for planning is available. A time limit of three minutes was regarded reasonable for planning. All grid jobs, which will be started within this time slot according to the old schedule, are regarded *fixed jobs* and will not become subject of rescheduling.

In section 2 a formal definition of the problem, a classification, and a comparison with other scheduling tasks will be given. Section 3 will describe the used algorithms, especially the new heuristics, and give a summary of the work carried out so far. The results of the experiments for assessing the effect of the new rescheduling heuristics will be presented in section 4, which will also report about first investigations regarding the maximum possible load for a grid, the size of which is growing proportionally to the amount of grid jobs.

2 Problem Definition and Classification

A notation common to the scheduling literature [3,4] is used to facilitate comparisons with other scheduling problems. Given are a set M of resources, a set J of application jobs, and a set O of grid jobs. The n grid jobs of application job J_i are denoted $O_{i1}, ..., O_{in}$. The following functions are given:

- a precedence function $p : O \times O \rightarrow \{TRUE, FALSE\}$ for the grid jobs
- an assignment function $\mu : O \rightarrow \mathcal{P}(\mathcal{P}(M))$ from grid jobs to resource sets. $\mathcal{P}(M)$ is the power set of M. μ_{ij} is the set of all possible combinations of resources from M, which together are able to perform the grid job O_{ij}
- a function $t : O \times \mathcal{P}(M) \rightarrow \mathbb{R}$, which gives for every grid job O_{ij} the time needed for the processing on a resource set $R_{ij} \in \mu_{ij}$
- a cost function, $c : \mathbb{R} \times \mathcal{P}(M) \rightarrow \mathbb{R}$, which gives for every time $z \in \mathbb{R}$ the costs per time unit of the given resource set

Optimisation is done by choosing suitable start times $s(O_{ij}) \in \mathbb{R}$ and resource allocations $R_{ij} \in \mu_{ij}$. A valid solution must meet the following two restrictions:

1. All grid jobs are planned and resources are allocated exclusively:

$$\forall\, O_{ij} : \exists\, s(O_{ij}) \in \mathbb{R}, R_{ij} \in \mu_{ij} : \forall M_j \in R_{ij} :$$

$$M_j \textit{ is in } [s(O_{ij}); s(O_{ij}) + t(O_{ij}, R_{ij})] \textit{ exclusively allocated by } O_{ij}.$$

2. Precedence relations are adhered to:

$$\forall\, i,\ j \neq k : p(O_{ij}, O_{ik}) \Rightarrow s(O_{ik}) \geq s(O_{ij}) \ + \ t(O_{ij}, R_{ij})$$

A violation of the two following soft constraints is treated by penalty functions in such a way that the amount of time and cost overruns is considered as well as the number of application jobs affected.

1. All application jobs J_i have a cost limit c_i, which must be observed:

$$\forall\ i : c_i \geq \sum_{j=1}^{n_i} \int_{s(O_{ij})}^{s(O_{ij})\ +\ t(O_{ij},R_{ij})} c(z, R_{ij})\ dz$$

2. All application jobs J_i have due dates d_i, which must be adhered to:

$$\forall\ i : d_i \geq s(O_{in})\ +\ t(O_{in}, R_{in}) \quad \textit{where } O_{in} \textit{ is the last grid job of } J_i$$

The fitness calculation is based on the above-mentioned four objectives and an auxiliary objective described in [2]. Lower and upper estimations for costs and processing times are calculated in the first planning stage of GORBA described in the next section. Except for the utilisation rate the relative value rv_i of every criterion i is calculated based on its actual value $v_{i,act}$ relative to these limits:

$$rv_i = \frac{v_{i,act} - v_{i,min}}{v_{i,max} - v_{i,min}}$$

This makes the single values rv_i independent of the task on hand and results in a percentage-like range. These values are weighted and summed up, which yields the *raw fitness*. To avoid unwanted compensation effects the criteria are sorted singly or in groups according to priorities. The criteria of the highest priority always contribute to the sum, while the others are added if all criteria of the next higher priority fulfil a given threshold value. Weights and priorities are based on experience and aimed at a fair compromise between users and resource providers. The tuning of the suggested adjustment is left to the system administrator. If the two soft constraints are violated, the raw fitness is lowered to the *end fitness* by a multiplication by the corresponding penalty function, each of which delivers a factor between 0 and 1. Otherwise, end fitness and raw fitness are identical.

Generalising, this task contains the *job shop scheduling problem* as a special case. The extensions are co-allocation of heterogeneous and alternative resources of different performances and time-dependent availability and costs, earliest start times and due dates, parallel execution of grid jobs, and more than one objective. As our task includes the job shop problem, it is NP-complete. For this reason and because of the three minutes runtime limit, approximated solutions can be expected only.

A comparable problem could not be found in literature, see e.g. [3] and [4] for a comprehensive presentation of scheduling problems. This corresponds to the results of the literature review found in [5]. There, it is concluded that only few publications deal with multiple objectives in scheduling and, if so, they mostly deal with single machine problems. Within the grid domain some papers dealing with multi-criteria recently were published. In [6] it is reported that most of them deal with two criteria, like e.g. [7], and that in most cases only one criterion is really optimised, while the other serves as a constraint, see e.g. [8,9]. The approach from [9] uses matrix-like chromosomes, which is probably the reason why they can handle about 30 jobs within one hour only. Kurowski et al. [10]

use a modified version of the weighted sum for a real multi-criteria optimisation, but do not handle workflows. Summarising, we did not find a report about a comparable amount of resources and grid job organised in workflows subject to a global multi-criteria optimisation. Of course, a lot of publications focus on partial aspects of this problem. For instance, the well-known Giffler-Thompson algorithm [11,12] was extended to the given problem, but surprisingly produced inferior results than our heuristics [2] described below.

3 Algorithms of GORBA and First Results

GORBA [2,13] uses advanced reservations and is based on Globus toolkit 4 at present. It executes a two-stage planning process. In the first stage the data of new application jobs are checked for plausibility and a set of heuristics is applied that immediately delivers first estimations of costs and completion times. These results are also used to seed the start population of the subsequent run of the Evolutionary Algorithm (EA) GLEAM (Global Learning Evolutionary Algorithm and Method) [14].

Firstly, the old heuristics used for the tuning of GORBA reported in [2,13] are described, followed by the new ones for rescheduling. Firstly, a sequence of grid jobs is produced by one of the following three heuristic precedence rules:

1. *Shortest due time*: jobs of the application job with the shortest due time first
2. *Shortest grid job working time*: grid jobs with the shortest working time first
3. *Shortest working time of application job*: grid jobs of the application job with the shortest working time first

In the next step resources are allocated to the grid jobs using one of the following three resource allocation strategies (**RAS**):

RAS-1: Use the fastest of the earliest available resources for all grid jobs
RAS-2: Use the cheapest of the earliest available resources for all grid jobs
RAS-3: Use RAS-1 or RAS-2 for all grid jobs of an application job according to its time/cost preference

As every RAS is applied to each grid job sequence, nine schedules are generated. The new heuristics use the grid job sequence of the old plan for grid jobs, which are subject to rescheduling, i.e. all grid jobs which have not already been started or will be started within the next three minutes. The new grid jobs are sorted according to one of the three heuristic rules already mentioned and added to the sequence of old jobs, yielding three different sequences. Resources are allocated using the three RAS and again, nine more schedules are generated, but this time based on the old plan. The best of these eighteen schedules is the result of the first planning stage, while all are used to seed the subsequent EA run of the second stage.

The EA GLEAM already contains some genetic operators designed for combinatorial problems. They are summarised here only, due to the lack of space, and the intereste reader is referred to [14]. A chromosome consists of a sequence of segments, containing a sequence of genes, each of which represents a grid job.

The gene sequence determines the scheduling sequence described later. Apart from the standard mutation, which changes the sequence of genes by simply shifting one of them, GLEAM contains the movement of gene segments and the inversion of their internal order. As segment boundaries can be changed by some mutations, the segments form an evolvable meta structure over the chromosomes. Segment boundaries are also used for the standard 1- and n-point crossover operators, which include a genetic repair that ensures that every offspring does not lack genes in the end. The evolvable segmentation and its associated operators among others distinguish GLEAM from most standard EAs. Besides the grid job genes, each chromosome contains a special additional gene for the selection of the RAS. A schedule is constructed from the grid job genes in the sequence of their position within the chromosome as follows:

Step 1: The earliest start time of a grid job is either the earliest start time of its application job or the latest end time of its predecessors, if any.
Step 2: According to the RAS selected by the RAS gene, a list of alternatives is produced for every primary resource.
Step 3: Beginning with the first resources of the lists, the duration of the job is calculated and it is searched for a free time slot for the primary resource and its depending ones, beginning at the earliest start time of step 1. If no suitable slot is found, the resources at the next position of the lists are used.
Step 4: The resources found are allocated to the grid job.

In a first development phase of GORBA the incorporated algorithms were tested and tuned using four different benchmark scenarios for planning an empty grid. They reflect all combinations of small/large resource alternatives (sR, lR) and small/large grid job dependencies (sD, lD) and, together with four different loads, yielded a total of 16 benchmarks [15]. We use synthetic benchmarks because it is easier to ensure and steer dissimilarities. This investigation is based on the results reported in [2,13]: Beside the described coding we use phenotypic repair of possible violations of precedence rules of the grid jobs and as additional crossover operator the well-known OX operator reported by Davis [16].

4 Experimental Results for Fast Rescheduling

There are various reasons for rescheduling, of which the introduction of new application jobs is the most likely one. Others are job cancellations or terminations, new resources, resource breakdowns, or changes in the availability or prices of resources. The experiments are based on the most likely scenario of new application jobs and shall answer the following three questions:

1. Does rescheduling benefit from the old plan? If so, to which fraction of finished and new grid jobs?
2. How effective are the old and new heuristics and the subsequent EA run?
3. Up to which amount of grid jobs and resources does the EA improve the best heuristically generated schedule?

As the two benchmark scenarios based on large degrees of dependencies have turned out to be harder than those using small degrees [2,13], they are used here for the experiments. They are denoted *sRlD* and *lRlD* (small or large Resource alternatives / large Dependencies). As pointed out in [2] and [13], their time and cost limits were set so tightly that the heuristics could not solve them without violating these soft constraints. One criterion of the usefulness of the EA run was to find fitting schedules, which was achieved in most, but not all cases. In addition to this criterion, the end fitness values obtained were also compared for the new investigations.

For the experiments reported here, the only EA parameter tuned was the population size varying from 90 to 900 for the runs investigating the first two questions. For the last question, smaller populations also had to be used, as will be described later on. For every benchmark setting and population size, 50 runs were done and the results were averaged. Confidence intervals and t-tests were used to check the significance of differences at a confidence range of 99%.

For the first two questions, the two basic benchmark scenarios were used for the first planning, with 10 resources and application jobs consisting of 100 and 200 grid jobs, respectively. Eight rescheduling events were compared, which take place when 10 or 20% of the grid jobs are finished and new application jobs with 10, 20, 30, or 50% grid jobs (relating to the original schedule size) are added. This results in 32 benchmark settings. We concentrated on small amounts of already processed grid jobs, because this is more likely in practice and gives a chance for the old schedule to be useful. Otherwise, the situation is coming closer to the already investigated "new planning" situation.

Fig. 1 compares the results for all 32 benchmark settings. It must be mentioned that a certain variation in the resulting normalised fitness values is not relevant,

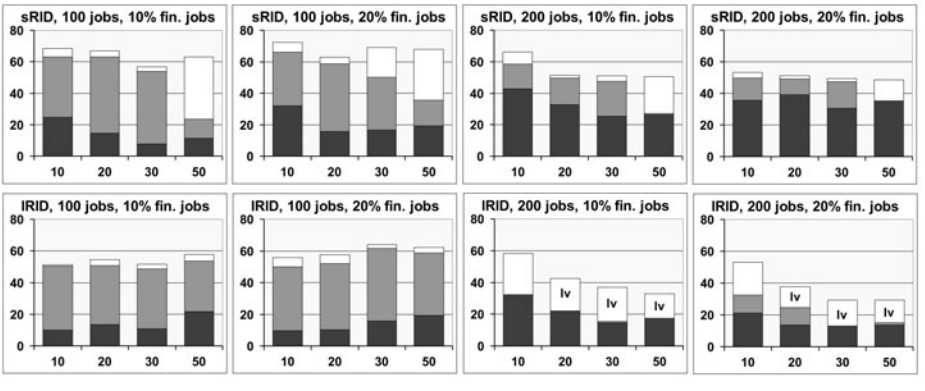


Fig. 1. Comparison of the fitness shares obtained from the basic heuristics (dark grey), rescheduling heuristics (light grey) and GLEAM (white) for all 32 rescheduling settings. X-axis: fraction of new grid jobs in percent relative to the original schedule size, y-axis: normalised end fitness. Abbreviations: *lv*: limit violations (mostly 1 to 3 application jobs violating the due date), for *sRlD* and *lRlD* see previous page.

Table 1. Comparison of the contributions of all rescheduling heuristics for different fractions of finished and new grid jobs. The best values of each column are marked dark grey, while values which reach 90% of the best at the minimum are marked light grey. Abbreviations: SWT: shortest work time, RAS: see section 3.

Finished grid jobs:	10%				20%				Ave-
New grid jobs:	10%	20%	30%	50%	10%	20%	30%	50%	rage
shortest due time & RAS-3	0.90	0.96	0.88	0.92	0.86	0.83	0.89	0.92	0.90
shortest due time & RAS-2	0.70	0.44	0.73	0.80	0.64	0.59	0.75	0.61	0.66
shortest due time & RAS-1	0.48	0.44	0.54	0.45	0.59	0.22	0.53	0.45	0.46
SWT of grid job & RAS-3	0.97	0. 86	0.81	0.69	0.94	0.78	0.81	0.62	0.81
SWT of grid job & RAS-2	0.74	0.42	0.63	0.53	0.66	0.50	0.68	0.39	0.57
SWT of grid job & RAS-1	0.47	0.41	0.46	0.28	0.57	0.24	0.54	0.26	0.40
SWT of appl. job & RAS-3	0.90	0.88	0.82	0.70	0.86	0.83	0.77	0.70	0.81
SWT of appl. job & RAS-2	0.70	0.41	0.70	0.56	0.64	0.51	0.57	0.46	0.57
SWT of appl. job & RAS-1	0.48	0.44	0.57	0.31	0.59	0.24	0.49	0.43	0.44

as the benchmarks are generated with some stochastic variations. Values between 50 and 70 may be considered good results. All GLEAM runs improve the fitness significantly. Even for the smallest improvement of benchmark *lRlD, 100 grid jobs, 10% fin. jobs*, the best heuristic fitness is clearly below the confidence interval of the EA result. The most important outcome is that for 10% new grid jobs, all eight scenarios perform well. The contribution of the heuristics is clearly situation-dependent and if they yield poor results, GLEAM compensates this in most cases. In other words, if the heuristics can solve the problem well, there is smaller room left for an improvement at all. Another nice result is that this compensation is also done to a certain extent for more new grid jobs, even if the schedules cannot be made free of limit violations. It can be expected that more new grid jobs will lower the contribution of the replanning heuristics and, in fact, this is confirmed by Fig. 1 for the instance of 50% new grid jobs. The case of *lRlD*, 200, and 10% finished grid jobs is somewhat exceptional, as the replanning heuristics do not work well even in the case of few new jobs.

Table 1 illustrates the contribution of each rescheduling heuristic. For each of the 32 benchmark settings, the fitness of each heuristic is calculated relative to the best heuristic result for this setting. The four values for both grid job amounts for *sRlD* and *lRlD* are averaged and shown in the table. The right column again averages the values for the different finished and new grid job fractions. The old heuristics based on short working times in the lower part of the table show the same poor behaviour as for planning an empty grid [2], but when taken as a basis for the new rescheduling heuristics, they contribute quite well. According to the table, RAS-3 performs best, but the raw material not shown here has thirteen cases, in which the two other RAS are the best ones. Thus, it is meaningful to use them all.

To investigate the third question, the rescheduling scenario with 10% finished and 10% new grid jobs is used with proportionally growing numbers for grid jobs and resources for the two basic benchmarks *sRlD* and *lRlD*. The comparison is

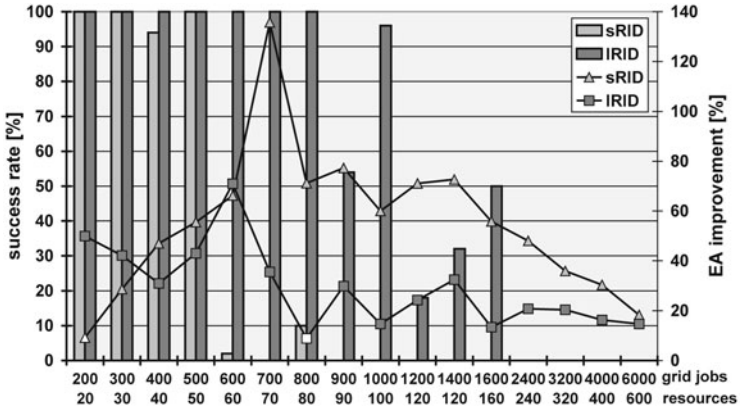


Fig. 2. Success rate and EA improvement compared to the best heuristic at increasing load for both basic benchmark scenarios and 10% finished and new grid jobs

based on the fitness improvement obtained by the EA compared to the best heuristic result and on the success rate, which is the ratio between violation-free and total runs per benchmark setting. Fig. 2 shows the results. As expected, success rate and EA improvement decrease with growing load. The relatively large variations of the EA improvement can be explained by the varying ability of the heuristics to produce schedules with more or less limit violations. As limit violations are penalised severely, a small difference already can produce a relatively large fitness alteration. No violations at all leave little room for improvements like in the two cases here marked by a white triangle or rectangle. The *sRlD* benchmark is harder to get violation-free, as shown by Fig. 2. This cannot be achieved in case of more than 500 grid jobs and 50 resources. If more resource alternatives are available, as it is the case for the *lRlD* benchmark, schedules free of limit violations can be achieved more easily and, in fact, this can be observed for up to 800 grid jobs and 80 resources. For greater loads up to 1600 grid jobs and 160 resources, there still is a chance of the schedules being free of violations. Starting from this load, the improvement rate is decreasing constantly. It may therefore be concluded that even for large loads like 6000 grid jobs and 600 resources, the EA delivers improvements below the level of schedules observing the limits completely. In other words, the amount of application jobs keeping the budgets is still increased compared to the heuristic results.

The more grid jobs must be planned, the less evaluations can be processed within the three minutes time frame. Fig. 3 shows that this amount decreases continuously with growing load. The more resource alternatives are available, the more must be checked by the RAS, which lasts longer and explains the lower numbers for the *lRlD* case. In the long run, the evaluations possible decrease to such an extent that the population size must be reduced to 20 or 30 to obtain two dozens of generations at least in the end. It is obvious that with such small

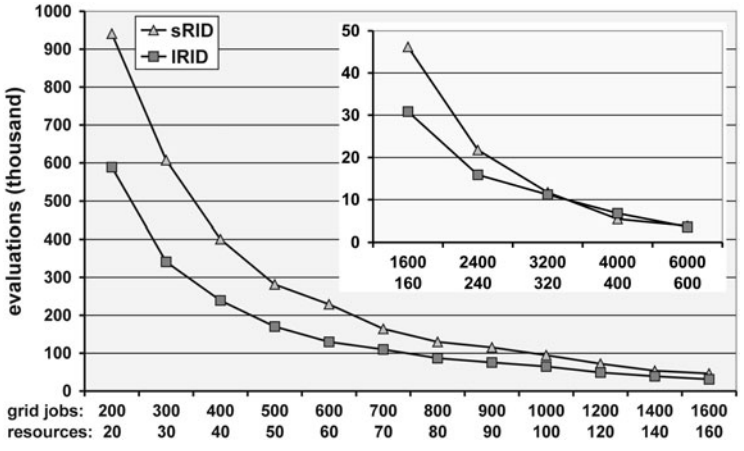


Fig. 3. Evaluations possible within the three minutes time frame at increasing load. For a better visibility, the diagram for 2400 and more grid jobs is shown separately.

numbers, only poor results can be expected and it is encouraging that even with the largest load there still is a small improvement. Hence, a faster implementation of the evaluation software will enlarge the possible load or deliver better results for the loads investigated.

5 Conclusion and Future Work

It was shown that the problem of scheduling grid jobs to resources based on realistic assumptions and taking the demands of resource users and providers into account is a much more complex task than just *job shop scheduling*. The task on hand enlarges the classical problem by alternative and heterogeneous resources, co-allocation, and last, but not least by multi-objective optimisation. The latter makes it hard to define a local searcher, as it is hard to decide whether a small local change of e.g. a job sequence is an improvement or not without setting up the complete allocation matrix. Consequently, we have not yet found a useful local searcher up to now. The investigated problems are rescheduling problems, which are the common case in grid resource management. Rescheduling is necessary, if new jobs arrive, planned ones are cancelled, resources break down or new ones are introduced, to mention only the more likely events. For this purpose, new heuristics that exploit the information contained in the "old plan" were introduced. It was shown that the solution for the common case of smaller changes, i.e. in the range of up to 20% finished and new grid jobs, could be improved significantly. The processible work load was also investigated for 10% finished and new grid jobs at an increasing number of jobs and resources. It was found that for loads of up to 6000 grid jobs and 600 resources, it was still possible to gain an improvement by the EA run within the given time frame of three minutes runtime.

References

1. Foster, I., Kesselman, C.: The Anatomy of the Grid: Enabling Scalable Virtual Organisations. Int. J. of Supercomputer Applications 15(3), 200–222 (2001)
2. Jakob, W., Quinte, A., Stucky, K.-U., Süß, W.: Fast Multi-objective Scheduling of Jobs to Constrained Resources Using a Hybrid Evolutionary Algorithm. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 1031–1040. Springer, Heidelberg (2008)
3. Brucker, P.: Scheduling Algorithms. Springer, Heidelberg (2004)
4. Brucker, P.: Complex Scheduling. Springer, Heidelberg (2006)
5. Setamaa-Karkkainen, A., Miettinen, K., Vuori, J.: Best Compromise Solution for a New Multiobjective Scheduling Problem. Comp. & OR 33(8), 2353–2368 (2006)
6. Wieczorek, M., Hoheisel, A., Prodan, R.: Taxonomy of the Multi-criteria Grid Workflow Scheduling Problem. In: Talia, D., et al. (eds.) Grid Middleware and Services - Challenges and Solutions, pp. 237–264. Springer, New York (2008)
7. Dutot, P.F., Eyraud, L., Mouni, G., Trystram, D.: Bi-criteria Algorithm for Scheduling Jobs on Cluster Platforms. In: Symp. on Par. Alg. and Arch., pp. 125–132 (2004)
8. Tsiakkouri, E., Sakellariou, S., Dikaiakos, M.D.: Scheduling Workflows with Budget Constraints. In: Gorlatch, S., Danelutto, M. (eds.) Conf. Proc. CoreGRID Workshop Integrated Research in Grid Computing, pp. 347–357 (2005)
9. Yu, J., Buyya, R.: A Budget Constrained Scheduling of Workflow Applications on Utility Grids using Genetic Algorithms. In: Conf. Proc. HPDC 2006. IEEE CS Press, Los Alamitos (2006)
10. Kurowski, K., Nabrzyski, J., Oleksiak, A., Weglarz, J.: Scheduling Jobs on the Grid - Multicriteria Approach. In: Computational Methods in Science and Technology, vol. 12(2), pp. 123–138. Scientific Publishers OWN, Poland (2006)
11. Giffler, B., Thompson, G.L.: Algorithms for Solving Production Scheduling Problems. Operations Research 8, 487–503 (1960)
12. Neumann, K., Morlock, M.: Operations Research. Carl Hanser, Mnchen (2002)
13. Jakob, W., Quinte, A., Süß, W., Stucky, K.-U.: Tackling the Grid Job Planning and Resource Allocation Problem Using a Hybrid Evolutionary Algorithm. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 589–599. Springer, Heidelberg (2008)
14. Blume, C., Jakob, W.: GLEAM – An Evolutionary Algorithm for Planning and Control Based on Evolution Strategy. In: Cantú-Paz, E. (ed.) GECCO 2002, vol. LBP, pp. 31–38 (2002)
15. Süß, W., Quinte, A., Jakob, W., Stucky, K.-U.: Construction of Benchmarks for Comparison of Grid Resource Planning Algorithms. In: Filipe, J., et al. (eds.) Conf. Proc. ICSOFT 2007, vol. PL, pp. 80–87 (2007)
16. Davis, L. (ed.): Handbook of Genetic Algorithms. V.N. Reinhold, New York (1991)

Comparison of Program Task Scheduling Algorithms for Dynamic SMP Clusters with Communication on the Fly

Łukasz Maśko[1], Marek Tudruj[1,2],
Gregory Mounie[3], and Denis Trystram[3]

[1] Institute of Computer Science of the Polish Academy of Sciences
ul. Ordona 21, 01–237 Warsaw, Poland
[2] Polish–Japanese Institute of Information Technology
ul. Koszykowa 86, 02–008 Warsaw, Poland
[3] Laboratoire Informatique et Distribution – IMAG
51 rue J. Kuntzman, 38330 Montbonot St. Martin, France
{masko,tudruj}@ipipan.waw.pl, {mounie,trystram}@imag.fr

Abstract. The paper presents comparison of the two scheduling algorithms developed for program structurization for execution in dynamic SMP clusters implemented in Systems on Chip (SoC) technology. SoC modules are built of a set of processors, memory modules and a multi–bus interconnection network. A set of such SoCs is interconnected by a global communication network. Inter–processor communication inside SoC modules uses a novel technique of data transfers on the fly. The algorithms present two different scheduling approaches. The first uses ETF–based genetically supported list scheduling heuristics to map nodes of a program to processors. The second is a clustering–based algorithm using Moldable Tasks (MT) to structure the graph. Both algorithms structure computations and local data transfers to introduce processor switching and data transfers on the fly. The algorithms were tested using a set of automatically generated parameterized program graphs. The results were compared to results obtained using a classic ETF–based list scheduling without data transmissions on the fly.

1 Introduction

The paper presents research results on scheduling for a new type of clustered SMP system [8]. This system is built of shared memory multiprocessor (SMP) modules implemented in a System on Chip (SoC) technology [4] connected via a global network. Each SoC module consists of a set of processors and shared memory modules, interconnected via a multi–bus local network. Processors are connected to a number of local memory busses and may be dynamically switched between them in runtime to form processor clusters. The local interconnection network provides data reads on the fly, which reduces the number of reads of the same data from shared memory to processor data caches for many processors connected to the same local memory module.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 31–41, 2010.

Program scheduling for parallel systems in a general case is an NP–complete problem. In the assumed architecture, apart from standard scheduling problems such as mapping of nodes to processors and data transfers to communication resources, one must also consider multi–level communication methods (transferring data in processor data caches between nodes, standard and on the fly data transfers via local memory buses and communication through a global interconnection network) and constraints on processor data caches. The paper presents experimental results with scheduling parameterized program graphs in the described architecture using two algorithms specially designed for such system. First of them is a 2-phase algorithm, based on list mapping. Its first phase considers mapping of tasks to processors located in different SoC modules and uses a genetic algorithm supported by a list scheduling with a modified ETF heuristics. The second phase transforms and schedules computations and communications to use processor switching and read on the fly facilities inside SoC modules. The second studied algorithm implements a clustering approach based on the notion of Moldable Tasks (MT), which are parallel tasks that contain some parallelism and may be executed on a different number of processors [3]. The program graph is first converted int MT–graph by dividing it into subgraphs of a special form, constituting MTs. Each MT is then scheduled for a range of numbers of processors, depending on the size of the SoC module in the target system. This scheduling includes introduction of reads on the fly and processor switchings to the MT graph. Finally, each task is alloted a number of processors and the whole MT graph is scheduled in the target system.

The paper cosists of 3 parts: first, the assumed system architecture is outlined. Then the scheduling algorithms are described. Finally, the experiments and comparison of the results obtained using the presented algorithms and a standard list scheduling algorithm with the ETF heuristics are presented.

2 Dynamic SMP Clusters Based on SoC Technology

Fig. 1a presents the general structure of the proposed system. Its basic elements are processors P and memory modules M. Fig. 1b presents the general structure of a sub-system that uses a local data exchange network to connect processors with memory modules. It includes a number of processors, a set of instruction memory modules, a set of data memory modules, a set of separate data and instruction caches and a set of local cluster networks to which processors (i.e. their data caches) can be connected. Each processor is equipped with many data cache modules, which provide multi-ported access to/from memory modules. Therefore, a processor can belong to many clusters at a time. This feature strongly improves data communication efficiency. All memory modules are also connected to the external peer to peer Global Network. Such structure can be implemented as a VLSI SoC module. A number of such modules can be connected via a global network such as memory bus or a crossbar switch. Special inter-processor synchronization hardware has to be included in the system to enable parallel execution of many synchronization operations for the program.

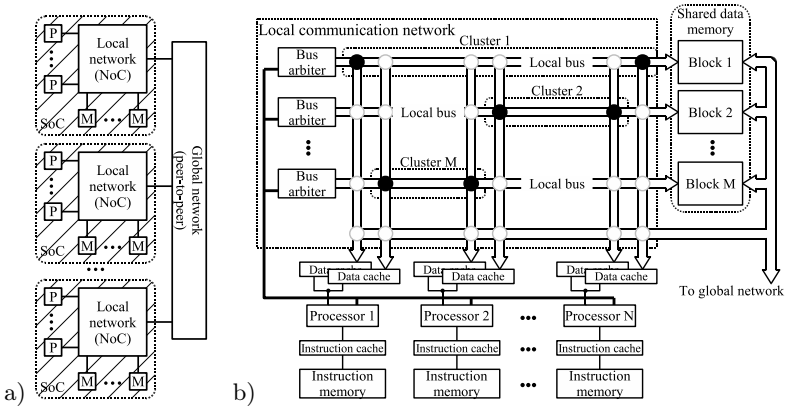


Fig. 1. General system structure a) and architecture of a single SoC module b)

To control communication in clusters: data pre-fetch, write, read on the fly and processor switching between clusters can be used. Reads on the fly are similar to cache injection. They consist in reading data on the fly from a memory module bus whose address lines are snooped by a special address snooping unit. Read on the fly requests are stored in the address snooping tables. Exact synchronization of a process that writes with reading ones is necessary. Requests such as memory write or read requests and synchronized on the fly read requests are serviced in processors by Communication Request Controllers (CRCs). Each memory module has an arbiter, which co-ordinates memory requests issued by CRCs. Processor switching between clusters consists in connecting a processor to a new cluster (i.e. its local network). A processor switched to a cluster can bring in its cache data, which are useful for the cluster. Other processors in this cluster can read data on the fly, when the switched processor writes then to the memory.

Tasks in programs are built in such way, that they do not require data cache reloading during their execution. All data have to be pre-fetched to processor data cache before a task begins. Current task results are sent to the cluster memory module only after task completes. This program execution paradigm, called cache–controlled macro data–flow principle, completely prevents data cache thrashing. The single assignment rule is used to avoid cache consistency problem in a case, when data produced by a task are to be modified by other parallel tasks.

More details on the architecture of the system can be found in [8].

3 Scheduling Algorithms

Program graph scheduling for the presented architecture deals with the following problems:

– Mapping of computing nodes of a program graph to processors in the executive system. It also includes mapping of processors to SoC modules.

– Mapping of data transfers to busses and memory modules in the system. Some of data transfers may already have these factors determined (if the communicating processors are mapped to different SoC modules, the read operation must use the global bus). For local data transfers this may include addition of processor switching between local busses to adjust connection configuration to local program needs.
– Analysis of local communication and its transformation to data transfers on the fly, wherever it is profitable.
– Analysis of processor data caches and transformation of a graph to such form, which obeys this constraint.

Optimization of data communication between data caches and memory modules has strong influence on the execution time of the scheduled programs. The proposed program transformations can be applied in a various order, giving various algorithms of a different quality. Also, for each of the above aspects, a different internal strategy may be used. The paper discusses two different approaches to program scheduling described in two consecutive sections of the paper.

3.1 2–Phase List–Based Scheduling Algorithm

The first presented algorithm takes advantage of a list scheduling method, adapting it to a situation, where a set of available processors is divided into subsets and communication between two processors may vary, depending on their placement [9]. The proposed algorithm consists of the two steps.

Distribution of program nodes between processors. At the beginning, all the computation nodes of the initial program graph must be distributed among all the processors in the system. In this step a standard macro dataflow notation of the program is used, in which program graph nodes correspond to computations and edges correspond to communications. The mapping is performed using a list scheduling algorithm with an ETF heuristics modified in such a way, that it respects the division of a set of all processors in the system to SoC modules. Processors are interconnected via a full network, in which the links between processors from the same SoC module are fast (local communication), while links between different SoCs are slow (global communication). Therefore, the global communications are penalized — their execution time is multiplied by the factor equal to the parameter `global_penalty`. The higher the value of this parameter, the more likely the nodes would be mapped to the processors in the same SoC module, even if it means their sequential execution. In this way, it is possible to optimally map communication–isolated subgraphs of the program graph, which should not be asigned to processors from different SoC modules.

A mapping routine is provided with the mapping vector $V = (a_1, \ldots, a_{MN})$, $0 \leq a_i < M$, where N is the number of processors in a SoC module and M is the number of SoC modules in the system. For each i, values in such a vector mean that a logical processor i is mapped to a SoC module a_i. The mapping is valid iff the the size of a SoC module is equal to N. To find the best mapping, the

optimal mapping vector V must be provided. This vector depends on a scheduled graph and its calculation requires theknowledge about the mapping algorithm. Therefore, in a general case, all the possible vectors should be examined to find the best one. For a given number M of SoC modules and N processors in a module, the number of all the possible mapping vectors equals $\frac{(MN)!}{M!N!^M}$. This number grows rapidly as the number of SoC modules or processors per module increases. Therefore the heuristic search algorithm must be used. To determine the best mapping vector, we use a genetic algorithm. Each chromosome defines one mapping vector (corresponds to one processor distribution scheme). It is a sequence of MN integers $(a_1 \dots a_{MN})$, where a_i denotes the SoC module to which processor i is mapped. According to such placement, for every pair of processors i and j, if $a_i \neq a_j$ all communication between these processors is done through the global interconnection network. If $a_i = a_j$, then communications between these processors use local communication networks, which implies, that in the future they can be executed using data transfers on the fly.

For each chromosome, the scheduled program graph is mapped to processors using a list scheduling algorithm with ETF heuristics adopted to obey the constraints derived from the mapping chromosome. The value of a fitness function Fit for a chromosome C depends on the execution time of nodes in so scheduled program graph and is determined by the following formula: $Fit(C) = const - \frac{\sum_{v \in E} end(v)}{|E|}$, where G is the graph scheduled according to the mapping given by a chromosome C, E is the set of nodes in G, which have no successors, $end(v)$ is the completion time of the node v and *const* is a constant, such that $Fit(C) > 0$ for every possible C. The presented fitness function promotes such chromosomes (from the ones which give the same graph execution time), which are *partially correct*. It means, that some parts of the graph are mapped optimally, while others may be not. The crossover and mutation operators consist in remapping random processors from one SoC module to another. If it is necessary, a chromosome is fixed by redistribution of processors from the SoC modules, which are too big, among other SoC modules according to the Best Fit rule to preserve as many processors as possible in the same module.

Structuring a mapped program graph. The main goal of the second phase is communication structuring. It aims at transforming local data transfers to reads on the fly. It includes conversion of kinds of nodes (standard reads to read on the fly), adding extra nodes to the graph (barriers, processor switchings). This process is iterative and based on local subgraphs corresponding to a communication schemes such as broadcasts, reductions or a butterflys (many–to–many communication), called basic structures [5]. Such transformations introduce synchronization, which is necessary for proper execution of data reads on the fly. Barriers impose delays in program execution or may lead to deadlocks. Therefore, in some cases it is necessary to exclude some nodes from a transformation. The nodes from the graph G are chosen according to times determined during simulated execution.

Finally, the graph is tuned to include the last constraint — the size of the processor data caches. The nodes are examined in the order determined by a

simulated execution of the graph. The algorithm calculates the data cache occupancy for each processor. If in any time point the required cache size exceeds the available cache capacity, the conflict is resolved so as to preserve the cache–driven macro dataflow program execution principle. Program graph transformations are automatically introduced which introduce additional steering edges, writes from processor data caches to a shared memory and reloading of this data back to processor data caches, when it is required.

3.2 MT–Based Scheduling Algorithm

The second discussed algorithm [6] works in three phases. First, the clustering is used to divide a graph to subgraphs, constituting Moldable Tasks. Then, each subgraph is analyzed separately. It is scheduled for a number of processors, using a list scheduling principles similar to the scheduling in the first algorithm. Then, for each MT, a number of processors is alloted and all the MTs are asssigned to processors in a target system.

A program graph is divided into subgraphs (G') of a limited size, which constitute Moldable Tasks created according to the following rules:

- For each node $v \in G'$, if G' contains any direct predecessor u of v, it contains all direct predecessors of v.
- For each node $u \in G'$, if G' contains any direct successor v of u, it contains all direct successors of u.
- For each pair of nodes $u, v \in G'$, if there exists a path between u and v in the graph, all the nodes on this path belong to G'.

The division is performed using clustering with a criterion based on a critical path CP (i.e. the heaviest path) of the graph. In every step, one edge from the current CP graph is selected. Then, the algorithm tries to merge the two MTs, which are connected with the selected edge. The merge operation includes graph closure operation, that follows the rules described above. If the newly created graph is not too big, it is accepted, otherwise it is rejected. The procedure continues until there are no unexamined edges (when all the edges on the CP are examined, also others are taken into account).

Each MT subgraph is then structured for a number of processors, from 1 to the number of processors in the SoC module. The structuring is performed in a similar way to the one in the previous algorithm (the nodes are mapped to the processors using a standard list scheduling, then communication and processors caches utilization is examined). Each execution time is remembered.

In the final step, each MT is alloted a number of processors, on which it will be executed and the tasks are assigned to processors in a target system. This process starts with a set of free MTs, i.e. the ones that have no predecessors in the MT–graph. They are communication–independent which means, that they may be executed in any order. For this set, each combination of possible allotment (assignment of the number of processors to a task) is examined and for each such allotment they are scheduled in a target system. Execution time of

each task depends on the number of alloted processors and is equal to the time determined in the previous step. The best allotment is chosen. After these tasks are examined, their successors constitute a set of tasks without uncompleted predecessors. Therefore the processors may be alloted to them in the same way as to their predecessors. This process is iterated, until there are no unvisited tasks. As a result, the program graph is structured and scheduled.

4 Experimental Results

As a testbed, a set of automatically generated *semi–regular* program graphs was used. The graphs consisted of 4 communication-intensive subgraphs executed in parallel. Each subgraph contained 7 layers of nodes, 3 to 4 nodes (randomly chosen) in each layer, so they had width equal to 16. The communication inside a subgraph took place only between consecutive layers. Additional rare random communication between the subgraphs can was introduced (examined 0 and 25 such communication edges per graph). The whole randomly generated exemplary graph is presented in Fig. 2. The communication–intensive elementary subgraphs are denoted by ovals. The bold edges represent additional inter–subgraph communication. Such graphs represent a higher level of difficulty then majority of graphs for real algorithms (usually regular).

All the computing nodes had the same weights (equal to 100). In the experiments we have assumed the following variable parameters: the input degree of graph nodes (two variants: 1–2, 3–4), the weight of communication edges (data volume: 25, 50, 100), the number of SoC modules and processors in the system (1 to 4 SoC modules, up to 16 processors in total). The system contained twice as many shared memory modules and local busses as the number of processors. The size of processor data cache depended on the scheduled program graph and was the same for each tested hardware configuration. For a given graph, the processor data cache volume was set to twice the maximal data size required by any of graph nodes. A set of graphs generated using these rules was automatically scheduled using the implementation of the presented algorithms. We have also examined a variant of these algorithms, when reads on the fly were not intro-

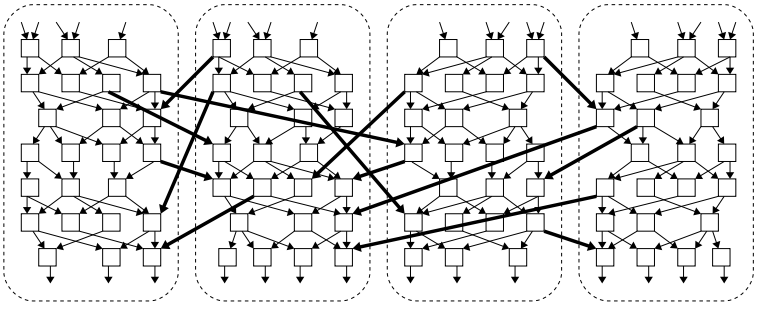


Fig. 2. A structure of an exemplary program graph

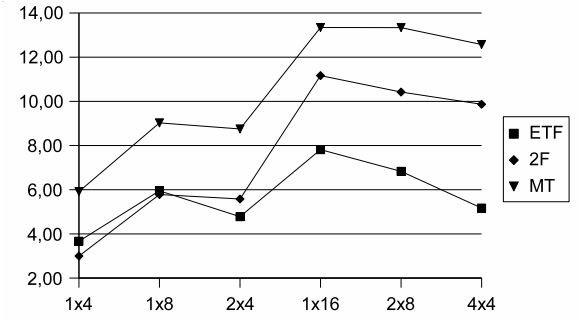


Fig. 3. Comparison of speedups obtained using the standard ETF scheduling, 2-phase with reads on the fly and MT-based with reads on the fly algorithms

duced to the program graph, to check the influence of reads on the fly on the execution time of graphs. The last scheduling algorithm (for comparison) was a classical list scheduling with the ETF heuristics, without application of reads on the fly, but with processor switching (due to mapping of transmissions to memory busses in a multi-bus SoC network). All the experiments were performed using a cycle–acurate software simulator written in C++.

There were 6 parallel configurations tested: 1 SoC module with 4 processors (1x4), 1 SoC module with 8 processors (1x8), 2 SoC modules with 4 processors each (2x4), 1 SoC modules with 16 processors (1x16), 2 SoC modules with 8 processors each (2x8) and 4 SoC modules with 4 processors in each (4x4). We have also evaluated the execution time of each graph on a single processor. Sequential execution obeyed the same constraints as the parallel one, including limited processor data cache size. For each configuration, the communication–to–computation speed ratio was set to 4. It meant that execution of a communication node with a given weight lasted 4 times longer then execution of a computation node with the same weight. The size of processor data cache for each simulated graph was set to twice the the largest requirement of a node in a graph (which is enough to execute a single thread without unnecessary data reloads and with some additional space for data transfers in processor data cache) and was the same for each hardware configuration, including sequential execution.

Fig. 3 presents the speedups obtained using a standard ETF-based list scheduling algorithm (ETF) compared to results obtained with presented algorithms. The 2-phase scheduling algorithm (2F) gives a bit worse results than the MT-based algorithm (MT). It is caused by the different approach to scheduling of by both algorithms. Like each list scheduling, the 2-phase algorithm prefers mapping parallel nodes to separate processors (if the communication doesn't disturb) rather than placing them on the same processor. If the width of the graph exceeds the number of available processors, nodes from different (potentially not related) parts of the graph are mixed on the same processor, reducing usage of data transfers through processor's data cache. MT-based scheduling algo-

rithm allows placing of parallel nodes on the same processor, but such nodes are selected from the same subgraphs of the scheduled graph. It increases the probability, that more data can be transferred via processor's data cache, improving execution time.

For configurations with 4 and 8 processors, the results obtained with a classic ETF-based algorithm were similar to the results of the 2–phase algorithm and both give worse results then the MT–based algorithm. This is due to the fact, that the serialization, which is necessary in this case, strongly decreases the posibility of using reads on the fly. The superlinear speedups obtained for graphs scheduled with the MT-based algorithm are caused by better utilization of processor data caches, which is not disturbed by threads belonging to different, not related parts of the graph. In configurations with the number of processors equal to the width of a graph, we may observe, that both proposed algorithms outperform the standard ETF-based scheduling. The results obtained with the MT and 2-phase algorithms were even 2.4 (1.9) times better. The decrease of performance in the case of 16 processors divided to 2 and 4 SoC modules (2x8, 4x4), regardless the algorithm, is caused by the global communication implied by the system structure composed of many SoC modules.

The graphs presented in Fig. 4 show the influence of reads on the fly on speedups obtained by both presented algorithms. When the number of processors is small, each computing unit must execute nodes from more than 1 parallel thread. This increases requirements on the size of processor data caches and forces the processor to send some data to shared memory before read on the fly transmission can take place. It makes such transmissions hardly possible or not optimal, therefore their presence has small influence on program execution time. Reads on the fly have a very impressive influence on execution speedups, when the executive system is equipped with the number of processors equal to the width of a scheduled graph (16 processors). In this case, each processor executes the nodes only from one thread, therefore the requirements on the size of processor data caches are weaker. In such situation there is no harmfull data cache reloading. Therefore data transmissions on the fly are much more effective

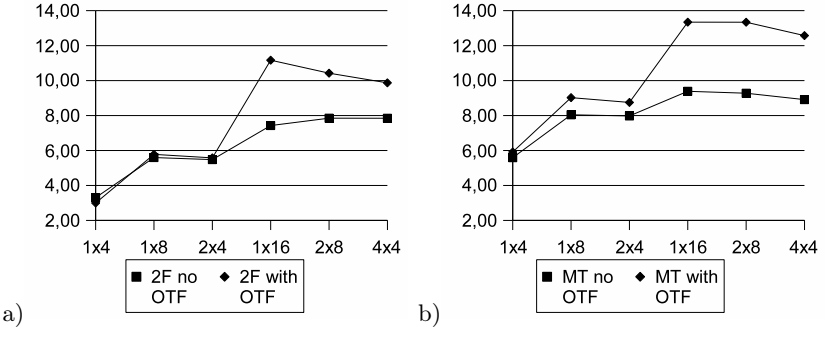


Fig. 4. Influence of reads on the fly (OTF) on speedups obtained using 2-phase (a) and MT-based (b) algorithms

in this case. The speedups due to reads on the fly reached 50% for 2-phase algorithm in a 1x16 configuration and over 43% for MT-based algorithm in 2x8 configuration. When the system is built of more than 1 SoC module (2x8, 4x4), the results obtained by our algorithms without reads on the fly are still better than the speedups for standard ETF-based list scheduling algorithm.

5 Conclusions

The paper describes two new algorithms for scheduling parallel programs for SoC dynamic SMP clusters with communication on the fly. The first uses list scheduling for mapping nodes to processors. Then it analyzes and transforms communication between processors, obeying processor cache size constraint. The second algorithm uses clustering to divide a graph to subgraphs (Moldable Tasks). Then, each subgraph is analyzed separately. It is scheduled for a number of processors, using a list scheduling similar to the scheduling in the first algorithm. Then, for each MT, a number of processors is alloted. Finally, all the MTs are asssigned to processors in a target system. The two algorithms are based on different assumptions, therefore they produce solutions of a different quality. The experimental results show, that they produce good results. They outperform a classic list scheduling algorithm with ETF heuristics, even when no reads on the fly are used. The results prove, that the algorithms are able to utilize all the features of the presented architecture, including reads on the fly and processor switching.

References

1. Hwang, J.-J., Chow, Y.-C., Anger, F.D., Lee, C.-Y.: Scheduling precedence graphs in systems with interprocessor communication times. SIAM Journal on Computing 18(2) (1989)
2. Leung, J.Y.-T., Anderson, J.H.: Handbook of scheduling. In: Algorithms, Models and Performance Analysis. Chapman and Hall, Boca Raton (2004)
3. Lepere, R., Trystram, D., Woeginger, G.J.: Approximation algorithms for scheduling malleable tasks under precedence constraints. In: 9th Annual European Symposium on Algorithms, LNCS. Springer, Heidelberg (2001)
4. Benini, L., De Micheli, G.: Networks on Chips: A New SoC Paradigm. IEEE Computing, 70–78 (Janyary 2002)
5. Maśko, Ł.: Atomic operations for task scheduling for systems based on communication on–the–fly between SMP clusters. In: 2nd International Symposium on Parallel and Distributed Computing, ISPDC 2003, Ljubljana, Slovenia. IEEE CS Press, Los Alamitos (2003)
6. Maśko, Ł., Dutot, P.–F., Mounié, G., Trystram, D., Tudruj, M.: Scheduling Moldable Tasks for Dynamic SMP Clusters in SoC Technology. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 879–887. Springer, Heidelberg (2006)

7. Tchernykh, A., et al.: Two Level Job–Scheduling Strategies for a Computational Grid. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 774–781. Springer, Heidelberg (2006)
8. Tudruj, M., Maśko, Ł.: Parallel Matrix Multiplication Based on Dynamic SMP Clusters in SoC Technology. In: Thulasiraman, P., He, X., Xu, T.L., Denko, M.K., Thulasiram, R.K., Yang, L.T., et al. (eds.) ISPA Workshops 2007. LNCS, vol. 4743, pp. 375–385. Springer, Heidelberg (2007)
9. Maśko, Ł., Tudruj, M.: Task Scheduling for SoC-Based Dynamic SMP Clusters with Communication on the Fly. In: Proceedings of the ISPDC 2008 Conference, Kraków, Poland. IEEE CS Press, Los Alamitos (2008)

Study on GEO Metaheuristic for Solving Multiprocessor Scheduling Problem

Piotr Switalski[1] and Franciszek Seredynski[2,3]

[1] University of Podlasie, Computer Science Department,
3 Maja 54, 08-110 Siedlce, Poland
peter@ii.ap.siedlce.pl
[2] Polish-Japanese Institute of Information Technology,
Koszykowa 86, 02-008 Warsaw, Poland
[3] Institute of Computer Science, Polish Academy of Sciences,
Ordona 21, 01-237 Warsaw, Poland
sered@ipipan.waw.pl

Abstract. We propose a solution of the multiprocessor scheduling problem based on applying a relatively new metaheuristic called Generalized Extremal Optimization (GEO). GEO is inspired by a simple coevolutionary model known as Bak-Sneppen model. The model describes an ecosystem consisting of N species. Evolution in this model is driven by a process in which the weakest species in the ecosystem, together with its nearest neighbors is always forced to mutate. This process shows characteristic of a phenomenon called a punctuated equilibrium which is observed in evolutionary biology. We interpret the multiprocessor scheduling problem in terms of the Bak-Sneppen model and apply the GEO algorithm to solve the problem. We compare GEO algorithm with well-known Simulated Annealing (SA) algorithm. Both algorithms have some similarities which are considered in this paper. Experimental results show that GEO despite of its simplicity outperforms SA algorithm in all range of the scheduling instances.

1 Introduction

Presently there exist many optimization problems in science and engineering [12] which are difficult to solve. These problems are often NP-complete problems [6], for which no efficient solutions have been found. NP-complete problems can be only solved approximately by existing techniques like randomization, parametrization or using heuristics (metaheuristics). Most methods based on local search algorithms applied to solve such complex problems, characterized often by multiple local optima, converge usually to local minima [5].

A more general approach is to use a global search algorithm. In this case, we can find a global optimum, but it requires a higher cost, e.g., computational time for solving optimization problems. One of the classes of the global optimization algorithms particularly worth considering are nature-inspired algorithms based on natural phenomena. These algorithms are based on observation of natural

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 42–51, 2010.

processes which are frequently self-optimized. The most commonly used algorithms of this class are Genetic Algorithms (GA) [7], Simulated Annealing [9], Particle Swarm Optimization algorithm [8], and Artificial Immune Systems [4]. Many of them have been recently applied to solve different variants of scheduling problem in the context of multiprocessor systems, cluster systems or grid computing (see, e.g., [15, 17, 2, 18]).

The multiprocessor task scheduling problem considered in the paper is one of NP-complete problems. The problem is a key factor for a parallel multiprocessor system to gain better performance. The objective of scheduling is usually to minimize the completion time of a parallel application consisted of a number of tasks executed in a parallel system [10]. In this problem the parallel application is represented by a Directed Acyclic Graph (DAG). Tasks of the application must be allocated into a multiprocessor system and a schedule of their execution in the system minimizing the completion time should be found. An optimal solution of the problem is difficult to find because of NP-hard nature of the problem. As mentioned above, different metaheuristics have been applied to solve the problem, however, the performance of these algorithms is still an open research problem and is the subject of current intensive study.

In this paper, we propose a relatively new metaheuristic called GEO [14] to solve the scheduling problem. Results of the experimental study show that the algorithm is very efficient in solving the problem and provides better results than SA, either for deterministic or random instances of the scheduling problem.

The paper is organized as follows. In the next section, we describe the multiprocessor scheduling problem. Section 3 presents the concept of GEO algorithm and its application for the scheduling problem. In Section 4 we present SA-based scheduling algorithm. Next, in Section 5 we show experimental results and compare GEO with SA algorithm. The last section contains conclusions.

2 Multiprocessor Scheduling

The multiprocessor scheduling problem is defined as follows (see, e.g.,[13]). A multiprocessor system is represented by an undirected unweighted graph $G_s = (V_s, E_s)$, called a *system graph*. V_s is the set of N_s nodes of the system graph representing processors with their local memories of a parallel computer of MIMD architecture. E_s is the set of edges representing bidirectional channels between processors and defines a topology of the multiprocessor system. Fig. 1a shows an example of a system graph representing a multiprocessor system consisting of two-processors P_0 and P_1. It is assumed that all processors have the same computational power and communication via links does not consume any processor time.

A parallel program is represented by a weighted directed acyclic graph $G_p = (V_p, E_p)$, called a *precedence task graph* or a *program graph*. V_p is the set of N_p nodes of the graph representing elementary tasks, which are indivisible computational units. There exists a precedence constraint relation between the tasks k and l in the precedence task graph if the output produced by task k has to be

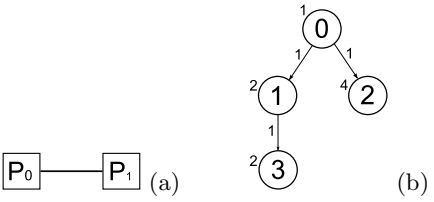


Fig. 1. Examples of the program and the system graphs: the graph of two-processor system in FULL2 architecture (a), an example of a program graph (b)

communicated to the task l. A program graph has two attributes: weights b_k and weights a_{kl}. Weights b_k of the nodes describe the processing time (computational cost) needed to execute a given task on any processor of a given multiprocessor system. E_p is the set of edges of the precedence task graph describing the communication pattern between the tasks. Weights a_{kl} of the edges describe communication time (communication cost) between pairs of tasks k and l when they are located in the neighboring processors. If the tasks k and l are located in the same processor, then the communication delay between them will be equal to 0. In the other case, the communication cost is proportional to the shortest distance between the processors i and j and it is equal to $a_{kl} * hops_{ij}$, where $hops_{ij}$ is a number of direct links of the shortest path in G_s between nodes (processors) i and j. Fig. 1b shows an example of the program graph consisting of four tasks with their order numbers from 0 to 3. All communication costs of the program graph are equal to 1 (see marked edges). Computational costs of tasks (marked on their left side) are 1, 2, 4, and 2, respectively. The purpose of scheduling is to distribute the tasks among the processors in such a way that the precedence constraints are preserved, and the response time T is minimized.

3 The Generalized Extremal Optimization Algorithm

3.1 Bak-Sneppen Model and Its Representation in Scheduling Problem

The idea of this algorithm is based on the Bak-Sneppen model [1]. Evolution in this model is driven by a process in which the weakest species in the population, together with its nearest neighbors, is always forced to mutate. The dynamics of this extremal process show characteristics of Self-Organized Criticality (SOC), such as punctuated equilibrium, that are also observed in natural ecosystems. Punctuated equilibrium is a theory known in evolutionary biology. It states that in evolution there are periods of stability punctuated by a change in an environment that forces relatively rapid adaptation by generating *avalanches*, large catastrophic events that effect the entire system. The probability distribution of these avalanches is described by a power law in the form

$$p_i = k_i^{-\tau}, \tag{1}$$

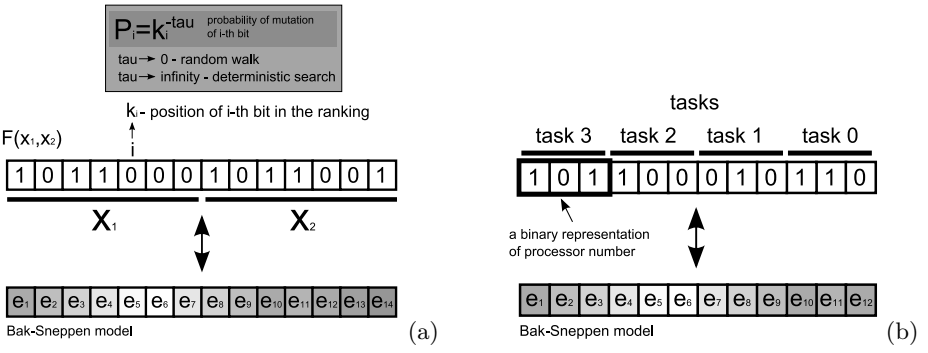


Fig. 2. Population of species in the Bak-Sneppen model and its correspondence in the GEO algorithm (a). Representation of the program graph and the system graph in Bak-Sneppen model (b).

where p_i is a probability of mutation of the i-th bit (species), k is a position of the i-th bit (species) in the ranking, τ is a positive parameter. If $\tau \to 0$, the algorithm performs a random search, while if $\tau \to \infty$, then the algorithm provides deterministic searching. Bak and Sneppen developed a simplified model of an ecosystem in which N species are placed side by side on a line. Fig. 2 shows the population of species in the Bak-Sneppen model and the idea of GEO algorithm [14]. In the GEO approach, a population of species is a string of bits that encodes the design variables of the optimization problem, and each bit corresponds to one species. In Fig. 2 a two variable function $F(x_1, x_2)$ is optimized. Each variable is coded using seven bits, and the whole string - a potential solution of the problem consists of 14 bits (upper part of Fig. 2(a)). Each bit of the string is considered as the species (lower part of Fig. 2(a)) of the Bak-Sneppen model. The number of bits per variable depends on the type of the problem. The population of the GEO algorithm to solve the multiprocessor scheduling problem contains a one n-bits string:

$$n = N_p * log_2 N_s, \tag{2}$$

where:

N_p - a number of tasks in a program graph, N_s - a number of processors. Fig. 2(b) (upper part) presents an example of the GEO string for the task graph scheduled into the multiprocessor system consisting of 8 processors. Groups of bits of the string represent processors to which corresponding tasks were allocated. One can see that, for example, the task 0 is allocated to processor 6, and task 3 to processor 5. The whole string consists of $n = 4 * log_2 8 = 12$ bits. Fig. 2(b) (lower part) shows a relation between coding used in the GEO to solve the scheduling problem and Bak-Sneppen model.

3.2 The GEO Algorithm

In this algorithm each bit (species) is forced to mutate with a probability proportional to its fitness. The fitness is a value associated with a given combination of bits of the GEO string, related to a problem presented in this study. Change of a single bit of the string results in changing its fitness and indicates the level of adaptability of each bit in the string. The fitness can increase or decrease if a bit is mutated (flipped). After performing a single changing of the string bits and calculating corresponding values of fitness function we can create the sorted ranking of bits by its fitness. From this moment on, the probability of mutation p_i of each i-th bit placed in the ranking can be calculated by Equation 1 described above. According to [14] the GEO algorithm can be described as follows:

1. Initialize randomly a binary string of length L that encodes N design variables of bit length equal to L/N.
2. For the current configuration C of bits, calculate the objective function value V and set $C_{best} = C$ and $V_{best} = V$.
3. For each bit i do the following,
   (a) flip the bit (from 0 to 1, or from 1 to 0) and calculate the objective function value V_i of the string configuration C_i,
   (b) set the bit fitness F_i as $(V_i - R)$, where R is a constant. It serves only as a reference number and can assume any value. The bit fitness indicates the relative gain (or loss) that one has in mutating the bit.
   (c) return the bit to its original value.
4. Rank the N bits according to their fitness values, from $k = 1$ for the least adapted bit to $k = N$ for the best adapted. In a minimization problem higher values of F_i will have higher ranking, and vice versa for maximization problems. If two or more bits have the same fitness, rank them in random order, but follow the general ranking rule.
5. Choose with an equal probability a bit i to mutate according to the probability distribution $p_i = k^{-\tau}$, where τ is an adjustable parameter. This process called a generation is continued until some bit is mutated.
6. Set $C = C_i$ and $V = V_i$.
7. If $F_i < F_{best}$ ($F_i > F_{best}$, for a maximization problem) then set $F_{best} = F_i$ and $C_{best} = C_i$.
8. Repeat steps 3 to 7 until a given stopping criterion is reached.
9. Return C_{best} and F_{best}.

4 The Simulated Annealing Algorithm

SA algorithm developed by Kirkpatrick [9] has been applied to solve a wide range of NP-hard optimization problems. Its basic idea comes from physics. In the SA method, each point S of the search space is analogous to a state of some physical system, and the function $C(S)$ to be minimized is analogous to the internal energy of the system in that state. The goal is to bring the system, from

an arbitrary initial state, to a state with the minimum possible energy. At each step of the simulation, a new state S' of the system is generated from the current state S by giving a controlled acceptance/rejection rule with probability p_{SA}:

$$p_{SA}(S \Rightarrow S') = \begin{cases} 1, & for C(S') < C(S) \\ exp(-\Delta E/k * TEMP) & \text{in the other case,} \end{cases} \quad (3)$$

where E is the energy, $TEMP$ is the temperature, and k is Boltzmann's constant. ΔE is described as $C(S') - C(S)$. This rule was proposed by Metropolis et al. and called Metropolis Transition [11]. We repeat each step with a slow decrease of temperature until the stop criterion is satisfied. Algorithm can be described by a pseudocode presented in Fig. 3. On the beginning we generate randomly an initial state S (solution of the problem) and estimate an initial temperature T_{INIT}. At the next step we generate a new state S' from state S. If objective function $C(S')$ is less than $C(S)$ then we accept a new solution S'. In the other case a new solution is accepted only with probability P_{SA}. We repeat this step in N iterations. After that the temperature $TEMP$ is decreased by a small value. The simplest and most common temperature decrement rule is:

$$TEMP_{i+1} = c * TEMP_i, \quad (4)$$

where c is a cooling constant close to, but smaller than 1. The algorithm stops when some specified stopping criterion is met e.g., when no improvement has been found for some number of moves.

GEO algorithm has some similarities to this algorithm. In GEO fitness of each bit F_i is determined by loss or gain when bit i is changed. In SA the loss or gain determines ΔE. In the next step of these algorithms a new solution is accepted with specified probability depends on actual temperature $TEMP$ and ΔE (SA algorithm), τ parameter and actual position of the bit k in the rank (GEO algorithm). In the SA algorithm the solution is also represented by a one string of bits, thus we use consistently the same coding of solution used by GEO.

```
start with an initial state S
set TEMP:=T_INIT (initial temperature)
set i:=0
repeat {
    set i:=i+1
    for j:=1 to N (predetermined constant) do {
        choose a neighbour state S' from S
        set DELTA:=C(S')-C(S)
        if (DELTA<0) or (random(0,1) < exp(-DELTA/TEMP))
           set S:=S'
    }
    TEMP:=c*TEMP (temperature reduction)
} until TEMP<LOW or END_TEST
```

Fig. 3. Pseudocode of the Simulated Annealing algorithm

5 Experimental Results

A number of experiments with deterministic program graphs known in the literature and random program graphs (see, e.g., [13]) have been conducted. The results were compared with those obtained with use of SA.

The probability p_i has significant influence on the convergence of GEO algorithm. A value of this probability depends intimately on the position of a mutated bit in the rank and the parameter τ. This parameter controls a range of potential bits to be mutated. Our previous work (see, [16]) was oriented on defining the right value of τ. For small program graphs (less than 100 tasks) only for $\tau = 1.25$ the algorithm provides an optimal solution. For a big program graphs (more than 100 tasks) as the optimal value of τ we assumed 1.75.

In the original form of the GEO algorithm only one bit is mutated in each generation. For small problems, where the string is short the mutation should be sufficient. Along with growing size of the population of bits, a mutation of one bit per generation can be not efficient. In [16] we can observed that for a bigger population (more than 200 species) mutation not a single bit, but mutation of three bits gives optimal results.

Calculation of the fitness function T is the main source of the time complexity of the considered algorithms. The complexity of an algorithm expressed by a number of the fitness function calculations depends not only on a number of tasks, but also on its topology and a relation between communication and computational costs in a given program graph. We assumed an equal number of evaluations of the fitness function for both algorithms. A number of evaluations is in range $N_E \in \{10000..500000\}$. We repeated each experiment 20 times.

In GEO, we used the following parameters in the experiments: $\tau \in \{1.25..1.75\}$, a number of mutated bits $N_b \in \{1..3\}$. For SA, we set a number of iterations $N \in \{50..150\}$, initial temperature $T_{INIT} \in \{30..100\}$, cooling constraint $c = 0.98-0.99$. At first, we conducted an experiment with use a two processor system (system graph: $FULL2$, random program graph: $g100\_10$). In Fig. 4 we show a typical run of GEO algorithm (a) and SA algorithm (b). One can see that

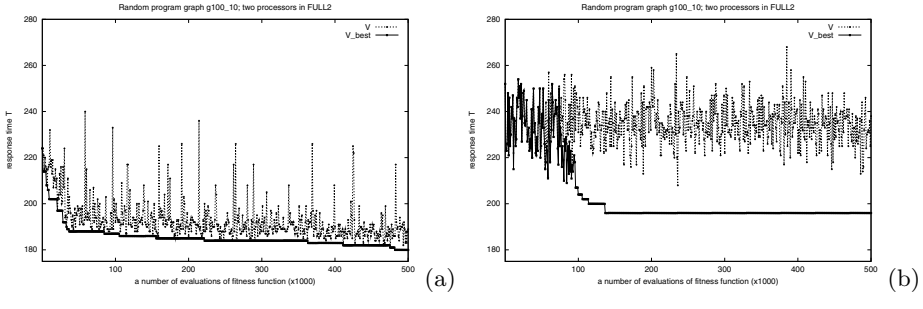


Fig. 4. Typical run of GEO (a) and SA (b) algorithm for the program graph $g100\_10$ and the system graph FULL2. The experiment was conducted with equal number of $N_E = 500000$ evaluations of fitness function.

SA indicates that the temperature $TEMP$ has significant impact on obtained results. On the first phase of the algorithm temperature is high, so the best result is highly diversified (for $N_E < 100000$). After reduces the temperature $TEMP$ the best result found by the algorithm is steadying. In the opposite to SA, GEO algorithm is finding the solution consistently and constantly. Simultaneously GEO obtains considerably better result than SA.

After that, we conduct regular experiments with use more than two processor system graphs ($N_s = 4$ and $N_s = 8$ processors). In this variant we are faced with the major problem - the length of population grows rapidly with a number of processors m. If we consider the program graph consisting of 406 tasks (*g400_5*) in the four processor system, the size of the population is equal to 406 * 2 = 812 (2 bits for coding of the four processors in a binary form). Table 5 presents results for the 4 and 8 processor systems. We analyzed difficult examples of program graphs: one deterministic graph *gauss18* and wide range of random graphs. As the first graph we used *gauss18*. For this experiment we assumed $N_E = 10000$. Because of small graph for GEO we set $\tau = 1.25$ and $N_b = 1$. The optimal response time T for four and eight processors is 44. GEO found the optimal solution. SA was noticeably worse. Afterwards we used random graphs (*g25*, *g100*, *g200* and *g400*). For these graphs we assumed $N_E = 500000$. These graphs are the most difficult instances of the problem for both algorithms. The experiments are summed up in Tab. 5. We conducted series of the experiments to find an optimal number N_b of bits to mutate and τ parameter. Table 5 shows optimal parameters: τ and N_b found for GEO algorithm for each the program

Table 1. The results of the algorithms: GEO and SA for random program graphs in the scheduling problem for $N_s = 4$ and $N_s = 8$ processors. The best response time T and the average of T (in rounded brackets) values on the base of 20 times. The best known results are in bold.

	$N_s = 4$			$N_s = 8$		
Program graph	**GEO**		**SA**	**GEO**		**SA**
	$\tau(N_b)$	$T(T_{avg})$	$T(T_{avg})$	$\tau(N_b)$	$T(T_{avg})$	$T(T_{avg})$
gauss18	1.25 (1)	**44**(45)	49(51)	1.25 (1)	**44**(47)	51(58)
g25_1	1.75 (1)	**305**(312)	334(342)	1.75 (2)	**289**(291)	304(309)
g25_5	1.75 (1)	**96**(97)	100(106)	1.75 (1)	**96**(102)	110(117)
g25_10	1.75 (1)	**62**(67)	95(103)	1.75 (2)	**62**(71)	112(119)
g100_1	1.75 (1)	775(787)	878(895)	1.75 (1)	582(591)	685(698)
g100_5	1.75 (3)	367(375)	415(426)	1.75 (3)	375(382)	412(413)
g100_10	1.75 (3)	170(174)	206(209)	1.75 (3)	184(192)	208(209)
g200_1	1.75 (1)	1543(1549)	1732(1755)	1.75 (3)	1059(1073)	1275(1293)
g200_5	1.75 (3)	451(457)	499(504)	1.75 (3)	422(426)	451(456)
g200_10	1.75 (3)	476(481)	506(510)	1.75 (3)	469(474)	486(488)
g400_1	1.75 (2)	3276(3286)	3605(3627)	1.75 (3)	2015(2035)	2398(2445)
g400_5	1.75 (3)	916(925)	1003(1008)	1.75 (3)	810(820)	869(877)
g400_10	1.75 (3)	506(508)	535(536)	1.75 (3)	481(486)	500(502)

graph and the system graph. Program graphs (*g25_1*, *g100_1*, *g200_1* and *g400_1*) required small changes, so the parameter N_b has value 1 or 2. Only for more complicated cases the value of this parameter is equal to 3. For other examples of these graphs N_b must be increased to 3. τ parameter has persistent value 1.75. Only for small graph *gauss18* this parameter is set to 1.25.

As we can notice, GEO can find appreciably better outcomes than SA. The differences are especially visible for graphs *g100*, *g200* and *g400*. Explanation of these results are consequence of behavior GEO and SA algorithms. GEO finding the solution by calculating fitness function for each bit from population and accepting population for which mutation of bit gave the best result. In SA we calculate fitness function only once for a current state S in a given iteration of the algorithm. In the next state S' there is a solution which no guarantees that will be better. In each step of GEO algorithm we can affirm that next solution will be better. Both algorithms measure lose or gain of the solution, but SA does it for following states which are generated randomly or by change a bit the previous state. In SA only Metropolis Transition control the converge of the algorithm. On the other side this method do not let to stop algorithm in local minimum. GEO finding a solution more consequently by precisely valuation of the bits of the population and choosing the most suitable population in the current step of the algorithm.

6 Conclusions

In this paper we have proposed a relatively new metaheuristic called GEO to solve the multiprocessor scheduling problem. It is based on the Bak-Sneppen model describing dynamics of ecosystems as the set of extremal processes known as Self-Organized Criticality. Applying the GEO algorithm to the task scheduling problem has confirmed that this algorithm is useful for scheduling problem. Simplicity of the GEO algorithm is one of its advantages. The performance of the algorithm depends in fact only on two parameters: the value τ and the number of mutated bits. These values were established experimentally for the scheduling problem.

We compared the results obtained by GEO with ones obtained by SA. In experiments deterministic and random program graphs were used. The results of the experiments show advantages of GEO over these well known metaheuristics. For simple program graphs all algorithms found optimal solutions, but even in this case GEO shows better average performance. SA seems to be similar for GEO. However, this algorithm cannot find satisfactory solutions for more complex program graphs due to mentioned disadvantages of SA.

References

[1] Bak, P., Sneppen, K.: Punctuated equilibrium and criticality in a simple model of evolution. Phys. Rev. Lett. 71, 4083–4086 (1993)

[2] Beham, A., Winkler, S., Wagner, S., Affenzeller, M.: Distributed, Heterogeneous Resource Management Using Artificial Immune Systems. In: Proc. of the 22nd IEEE International Parallel & Distributed Processing Symposiumm, NIDISC Workshop (2008)

[3] Bollobas, B.: Random Graphs, pp. 34–42. Academic Press, New York (1985)
[4] Dasgupta, D.: Artificial Immune Systems and Their Applications. Springer, Berlin (1999)
[5] Eldred, M.S.: Optimization Strategies for Complex Engineering Applications. Sandia Technical Report SAND98-0340 (1998)
[6] Garey, M.P., Johnson, D.S.: Computers and intractability - a guide to NP-completeness. W.H. Freeman and Company, San Francisco (1979)
[7] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
[8] Kennedy, J.: Swarm Intelligence. In: Zomaya, A.Y. (ed.) Handbook of Nature-Inspired and Innovative Computing, pp. 187–219. Springer, Heidelberg (2006)
[9] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing, vol. 220(4598), pp. 671–680. ACM, New York (1983)
[10] Kwok, Y., Ahmad, I.: Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. ACM Computing Surveys 31(4), 406–471 (1999)
[11] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of State Calculations by Fast Computing Machines. Journal of Chemical Physics 21(6), 1087–1092 (1953)
[12] Pardalos, P.M., Romeijn, H.E.: Handbook of Global Optimization, vol. 2. Springer, Heidelberg (2002)
[13] Seredynski, F., Zomaya, A.Y.: Sequential and Parallel Cellular Automata-Based Scheduling Algorithms. IEEE Trans. on Parallel Distributed Systems 13(10), 1009–1023 (2002)
[14] Sousa, F.L., Ramos, F.M., Galski, R.L., Muraoka, I.: Generalized Extremal Optimization: A New Meta-heuristic Inspired by a Model of Natural Evolution. In: Recent Developments in Biologically Inspired Computing, pp. 41–60 (2004)
[15] Swiecicka, A., Seredynski, F., Zomaya, A.Y.: Multiprocessor Scheduling and Rescheduling with use of Cellular Automata and Artificial Immune System Support. IEEE Trans. on Parallel Distributed Systems 17(3), 253–262 (2006)
[16] Switalski, P., Seredynski, F.: Generalized Extremal Optimization for Solving Multiprocessor Task Scheduling Problem. In: Li, X., Kirley, M., Zhang, M., Green, D., Ciesielski, V., Abbass, H.A., Michalewicz, Z., Hendtlass, T., Deb, K., Tan, K.C., Branke, J., Shi, Y. (eds.) SEAL 2008. LNCS, vol. 5361, pp. 161–169. Springer, Heidelberg (2008)
[17] Wilson, L.A.: Distributed, Heterogeneous Resource Management Using Artificial Immune Systems. In: Proc. of the 22nd IEEE International Parallel & Distributed Processing Symposium, NIDISC Workshop (2008)
[18] Xhafa, F., Alba, E., Dorronsoro, B.: Efficient Batch Job Scheduling in Grids using Cellular Memetic Algorithms. In: Proc. of the 22nd IEEE International Parallel & Distributed Processing Symposium, NIDISC Workshop (2007)

Online Scheduling of Parallel Jobs on Hypercubes: Maximizing the Throughput

Ondřej Zajíček[1], Jiří Sgall[2], and Tomáš Ebenlendr[1]

[1] Institute of Mathematics, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic
[2] Dept. of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Malostranské náměstí 25, CZ-11800 Praha 1, Czech Republic

Abstract. We study the online problem of scheduling unit-time parallel jobs on hypercubes. A parallel job has to be scheduled between its release time and deadline on a subcube of processors. The objective is to maximize the number of early jobs. We provide a 1.6-competitive algorithm for the problem and prove that no deterministic algorithm is better than 1.4-competitive.

1 Introduction

We consider scheduling of parallel jobs on hypercubes with the objective to maximize the number of jobs completed before their deadline. We focus on the case where all processing times are equal to 1. In this case, each job is specified by an integral release time and deadline, and the number of processors it needs, which is required to be a power of two, to respect the hypercube topology.

In the online setting, the jobs arrive over time: Each job arrives at its release time; at this time its complete specification is released. At each time step we need to choose a subset of available jobs that are scheduled. Available jobs are those that are already released, not yet scheduled, and with a deadline strictly larger than the current time. The total number of processors required by the chosen jobs needs to be at most the size of the hypercube.

The hypercube topology restricts the actual assignment of parallel jobs: The processors are organized as a hypercube and each job has to be scheduled on a subcube of the hypercube. However, since we consider only jobs with unit processing times, this restriction is equivalent to the constraint that job sizes as well as the total number of processors are powers of two. Once the total processor requirement is at most the number of processors, we can always assign the chosen jobs to subcubes in a greedy manner from the largest job to the smallest one.

Our results. We present a 1.6-competitive algorithm for this problem. In two special cases we show that the algorithm is 1.5-competitive. The first special case excludes jobs that require the whole hypercube. The second special case is that of tall/small jobs, where each job may require either the whole hypercube or a single processor. We show that the analysis of this algorithm is tight. Our algorithm is memoryless, i.e., its action at each time depends only on the currently available jobs.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 52–61, 2010.

We prove that no deterministic algorithm is better than 1.4-competitive. This is true even on a machine with two processors, which is a subcase of the tall/small special case.

Related results. If we restrict ourselves to sequential jobs (i.e., jobs requiring a single processor), the problem is trivial. The natural algorithm always schedules the jobs with the smallest deadlines (among the available jobs). A standard exchange argument shows that this is an optimal schedule. Once parallel jobs are introduced, this no longer works. We need to find a rule to choose, for example, between urgent parallel jobs and sequential jobs with large deadlines.

A simple approach to similar problems is the greedy algorithm. This works even if we allow both parallel jobs and weights. In each time step, we schedule a set of jobs with maximal total weight from the available jobs. A standard charging argument shows that this algorithm is 2-competitive. For each job in the optimal schedule, charge its weight to the timeslot in the online schedule where it is scheduled; if it is not scheduled, charge it to the same timeslot. To each timeslot in the online schedule, we charge at most twice the total weight of the jobs scheduled by the online algorithm: First, we may charge each job to itself. Second, we charge the jobs scheduled by the optimum at the same time, but not scheduled by the online algorithm; these jobs are available, thus their total weight is at most the weight of the jobs scheduled greedily. Summing over all the timeslots, 2-competitiveness follows. Improving the competitive ratio below 2 for this general problem is a challenging open problem.

The complexity of the offline problem is not known. Typically, parallel scheduling problems are NP-hard because they include some partitioning problem. Either partitioning the processors among the jobs, or partitioning the jobs into groups with the same total processing time. The hypercube constraint and the restriction to unit processing times make these packing problems trivial.

Nevertheless, polynomial algorithms are known only for a couple of special cases. One can maximize the number of completed jobs if all the release times are equal, see [3]. This was generalized to the case of nested intervals given by the release times and the deadlines, see [4].

For general release times and deadlines, the only positive result exists for the tall/small case studied in [1], see also [2] for an alternative proof; however, this gives only an algorithm for testing if all jobs can be completed. The throughput maximization is open even for the case of two processors, which is a special case of the tall/small case.

Preliminaries. The problem has a parameter m giving the number of machines. An instance of the problem consists of a set of n jobs. Each job J has an integral release time r_J, an integral deadline d_J and a size s_J (the number of requested processors). The numbers m and s_J are powers of two. As all times are integers and jobs' processing times are equal to one, instead of time we can consider timeslots (aligned unit-time intervals) and every job requests one timeslot.

We say that job J is feasible at timeslot T if $r_J \leq T$ and $T < d_J$. We say that job J is available at timeslot T if it is feasible and not scheduled yet. We say that job J is urgent at timeslot T if $d_J = T + 1$. A schedule assigns to each

processed job J find a timeslot T such that J is feasible at T, and s_J processors, so that no processor is assigned to two jobs at the same time. The objective is to find a schedule maximizing the number of processed jobs.

In the online variant of the problem, at the timeslot T, we get a knowledge of all jobs J with $r_J = T$ and we have to decide which jobs start to process at timeslot T.

We consider a variant of the generalized problem where all jobs have unit processing time ($p_J = 1$) and their release times and deadlines are integers. We also restrict the size s_J of jobs and the number of processors m to be a power of two. As mentioned in the introduction, this is equivalent to the requirement that each job is scheduled on a subcube of the hypercube with m processors.

We fix an ordering $\prec$ on jobs that is a strict linear ordering based on the ordering of deadlines, in a case of equal deadlines it is defined arbitrarily. For example, we take an ordering defined by formula $J_i \prec J_j \Leftrightarrow d_i < d_j \vee (d_i = d_j \wedge i < j)$. We suppose, w.l.o.g., that any algorithm chooses the $\prec$-minimal job from available jobs of the same size when it needs to choose one job of that size.

We use ALG to denote the analyzed algorithm and OPT to denote an optimal offline algorithm. Jobs of size m are called *max-jobs*, smaller jobs are called *non-max jobs*. Jobs of size 2^i are called *i-jobs* (where i is some number).

2 Algorithm

We want an algorithm that chooses from possible schedules according to these four rules, in the order of importance, because such rules lead to invariants used in the proof of the competitive ratio:

1. Prefer more smaller jobs over one bigger job.
2. Prefer an urgent job over a non-urgent job.
3. Prefer a bigger job over a smaller job.
4. Prefer $\prec$-minimal jobs among the jobs of the same size.

It is easy to convert these rules to a memoryless algorithm that (for each timeslot) examines a set of currently available jobs and chooses its maximal schedulable subset (a set such that the sum of the sizes of its members is less than or equal to m) satisfying these rules (e.g., if there is a non-urgent job in the chosen subset, then there is no urgent job of the same or smaller size outside of the chosen subset). The chosen subset will be scheduled in that timeslot.

Lemma 1. *The competitive ratio of the algorithm is at least* 1.6.

Proof. We construct an instance on four machines. (For more machines, it can be easily scaled up.) One 1-job X with deadline 3 and one 2-job A with deadline 5 are released at time 1. The algorithm chooses job A at time 1 (by rule 3). Consequently, two 1-jobs Y and Z with deadline 3 and four 0-jobs B, C, D, E with deadline 4 are released at time 2 and the algorithm chooses four 0-jobs (by rule 1) and loses all three 1-jobs (X, Y, Z). OPT schedules all jobs: three 1-jobs in the first two timeslots, four 0-jobs in timeslot 3 and the remaining 2-job in timeslot 4. The proof is summed up in Figure 1. □

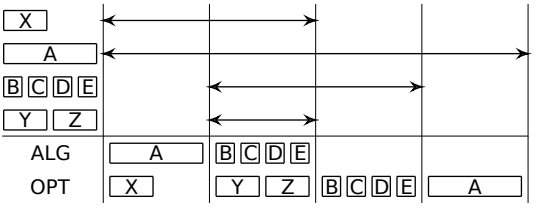


Fig. 1. The proof of the lower bound in a general case

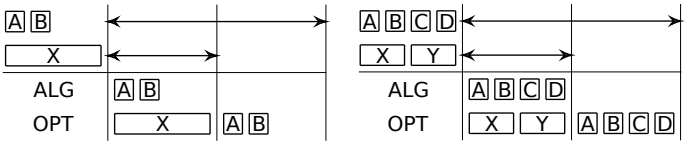


Fig. 2. The proof of the lower bound in the restricted cases

Lemma 2. *The competitive ratio of the algorithm is at least* 1.5 *in the tall/small case and in the non-max case.*

Proof. We construct two instances on four machines. Some urgent larger jobs and more non-urgent smaller jobs are released at time 1. The algorithm chooses more non-urgent smaller jobs and loses the urgent larger jobs. Details of sizes and counts of jobs are summed up in Figure 2, the left-hand side is for the tall-small case and the right-hand side is for the non-max case. The proof can be easily scaled up for more machines. □

3 Competitive Ratio

We prove the upper bound for the competitive ratio of ALG using a charging scheme. When we consider jobs in ALG and OPT schedules as two sets of vertices, then the charging scheme is a set of rules for a specification of weighted edges between these two sets to create a bipartite graph. This graph obeys some constraints: For each job in OPT schedule the sum of the weights of incident edges is exactly 1 and for each job in ALG schedule the sum of weights of incident edges is at most 1.6 (or 1.5 in the restricted cases). These constraints (and the fact that this scheme specifies such a matching for OPT and ALG schedules of every instance) imply that the competitive ratio of the algorithm is at most 1.6 (or 1.5 in the restricted cases).

We introduce some terminology. When there is an edge between two jobs with weight x we write that the job in OPT schedule *sends* x and the job in ALG schedule *receives* x. The charging scheme uses mainly two kinds of edges: *diagonal edges* and *vertical edges*. A diagonal edge is an edge from a job in OPT

schedule to the same job in ALG schedule in a different timeslot. A vertical edge is an edge from a job in OPT schedule to any job in ALG schedule in the same timeslot. A job not scheduled by ALG (but possibly scheduled by OPT) is called *an unscheduled job*. A job scheduled by OPT and not scheduled by ALG during that or earlier timeslots (but possibly scheduled later) is called *a free job* because it is available for ALG at the timeslot in which it is scheduled by OPT.

We use *a job* in two slightly different meanings. First, there is a particular job from an instance of a problem. Second, the job is scheduled by a particular schedule to some machines and some timeslot. The position occupied by some job in the particular schedule is also called the job. Specifically, we use *ALG-job* for the position of a job in ALG schedule and *OPT-job* for the position of a job in OPT schedule. Obviously, the charging edges do not connect jobs in the first sense, but ALG-jobs and OPT-jobs.

If there is a max-job in ALG schedule and in the same timeslot there is only one non-max free job in OPT schedule, then we call this non-max free job *a red job*. Other free jobs are called *white jobs*, non-free jobs (scheduled first by ALG and later by OPT) are called *black jobs*. In the first part of proof we define charging for white and black jobs, in the second part for red jobs.

The charging scheme is specified as follows: Each black job charges one diagonal edge (forwards to the same job in ALG schedule) and each white job charges one vertical edge (upwards to an unspecified job in the same timeslot). We will specify exact rules for a distribution of vertical edges to ALG-jobs later.

Matching of i-jobs at timeslot T is a process that finds a maximal matching between a set of i-jobs in ALG schedule of timeslot T and a set of white i-jobs in OPT schedule of timeslot T. If there is a job scheduled at timeslot T by both ALG and OPT, then it is matched with itself, remaining jobs are matched arbitrary with one restriction: any red jobs J in ALG schedule are matched at the end, only when no other jobs remain. Some i-jobs may be left unmatched in ALG or OPT schedule, but not in both schedules.

Lemma 3. *If an ALG-job A (scheduled at some timeslot T) is matched with OPT-job B, then A receives nothing diagonally (from OPT-job A).*

Proof. Suppose ALG-job A receives diagonally from (black) OPT-job A. Jobs A and B have to be different jobs, because OPT-job B is white. Because B is a white job, it follows that ALG did not schedule B before or at timeslot T. Because A is black, OPT scheduled A after timeslot T. Thus both A and B were available to both ALG and OPT at timeslot T, but ALG scheduled A and didn't schedule B and OPT scheduled B and didn't schedule A. This is a contradiction because A and B are jobs of the same size and both algorithms choose the $\prec$-minimal jobs from available jobs of the same size. □

Lemma 4. *For every timeslot it is possible to find a distribution of weight of all incoming vertical edges between ALG-jobs of the timeslot such that every job in ALG schedule can be categorized to at least one of these classes:*

- *Class C (common): The job receives at most* $1/2$ *vertically.*
- *Class M (matched): The job receives* 1 *vertically from the matched job.*

- *Class U (urgent): The job is urgent and receives at most* 1 *vertically.*
- *Class S (special): The job is scheduled at a timeslot that is full of jobs of the same size in ALG schedule. Furthermore, it is a non-max job and at most one job per timeslot can be the class S job. The job receives* 1 *vertically from the matched job and* 1/2 *vertically from another job.*

Proof. The proof is done independently for each timeslot. We show that for each white job in OPT schedule we find the same job or two other jobs in ALG schedule (in the same timeslot). Let T be any fixed timeslot. We use ALG_T (and OPT_T) schedule for ALG (and OPT) schedule restricted to timeslot T.

If there is no job in ALG_T schedule, then all jobs in OPT_T schedule have to be black, because any white OPT_T job could also be scheduled by ALG at T. So suppose there are some jobs in ALG_T schedule and the biggest job among them is an i-job. Jobs smaller than 2^i will be called small jobs. It is easy to see that there is no more than one small white job in OPT_T schedule—otherwise ALG should schedule two (or more) small jobs instead of the i-job. We distinguish two cases: one small white job and no small white job.

Case 1: There is exactly one small white job J in OPT_T schedule. First we match i-jobs in T. We split the timeslot in ALG_T schedule to slots of size 2^i. In each slot there is either one i-job or more small jobs (there is neither an empty slot nor a slot with one small job, otherwise the free space in that slot is large enough that ALG should schedule the job J in it). Now we assign those slots to OPT_T white jobs. The idea is that each OPT_T white i-job gets one slot and larger white jobs get proportionally more slots. Slots with matched i-jobs are assigned to matched OPT_T white i-jobs. If there are remaining OPT_T white i-jobs, they get slots with more small jobs. If we disregard job J then the rest is correct: matched ALG_T i-jobs are class M jobs, smaller jobs (assigned together to one job) are class C as well as remaining i-jobs assigned together to larger jobs. Unused ALG_T jobs may be class C as they receive nothing vertically. Now we find the assignment for job J. There are two cases:

Case 1.1: There is at least one slot with more small jobs. Then we assign it in the first place to job J (and those small jobs are class C) and the lemma holds.

Case 1.2: There are only i-jobs in ALG_T schedule (and one i-job called job K is assigned to job J). We have three cases distinguished by the structure of OPT_T schedule.

Case 1.2.1: There is at least one white i-job (job L) in OPT_T schedule. Then job L is matched with some ALG_T i-job (job L'). Job L' receives 1 vertically from job L; hence, it can receive additional 1/2 from job J and become a class S job. Additional constraints for a class S job also hold: L' is a non-max job, as otherwise ALG should have scheduled the two white jobs J and L by rule 1. There is only one class S job, because there is only one J job. Job K receives remaining 1/2 from job J, is a class C job and the lemma holds.

Case 1.2.2: There is no white i-job in OPT_T schedule but there are some larger white jobs. Then there are two unused slots in ALG_T schedule, because the sum

of sizes of larger OPT_T jobs is a multiple of 2^{i+1} and the number of ALG_T i-jobs assigned to them is even. Therefore, there are at least two ALG_T i-jobs available, they receive 1/2 from job J and are class C.

Case 1.2.3: Job J is the only white job in OPT_T schedule. If there are more than one ALG_T i-job then two of them receive 1/2 and are class C. If there is only one job M, then M has to be max-job, because there is no empty slot (ALG_T is full of i-jobs). This case cannot appear, as job J, which is not a max-job because it is a small job, would be a red job and not a white job.

Case 2: There is no small white job in OPT_T schedule. Let j-jobs be the smallest white OPT_T jobs, obviously $j \geq i$. First we match j-jobs (which does nothing if $j > i$). We split timeslot T in ALG_T schedule to slots of size 2^j. No such slot is empty (otherwise, ALG should schedule some white j-jobs scheduled by OPT at T). At most one slot is not full (because job sizes are powers of two we can always pack jobs from two half-empty slots to make one slot empty or full). Now we assign the slots to OPT_T white jobs as we did in the first case. If we have only slots with either one j-job or with more smaller jobs then it is the same argument as in first case (even easier because there is no job J). But the one non-full slot can contain only one job (job N), which is smaller than j-job. In that case job N has to be urgent: otherwise, ALG should schedule some white j-job instead of job N, by rule 3. Therefore, job N is a class U job and the slot with job N may be used much like a slot with two jobs. Even in this case the lemma holds. □

Lemma 5. *Let $I(J)$ (for any non-black job J) be a time interval starting by the timeslot when the time when job J was scheduled by OPT and ending by the last timeslot when J was available for ALG (it was scheduled by ALG or it was just before the deadline for unscheduled job J). Then $I(J)$ for all red jobs J do not overlap.*

Proof. Suppose two such intervals overlap. Let T be the first timeslot in their intersection. In timeslot T both jobs are available for ALG, ALG should schedule both jobs together (as they are non-max) but it scheduled one max-job instead. □

Lemma 6. *Let T be a timeslot when some unscheduled red job J is urgent. Then either there is an ALG job receiving at most* 1 *in timeslot T, or there are at least four ALG jobs in timeslot T.*

Proof. We split timeslot T to slots of the same size as job J. In each slot there is either one urgent job (or part of that job) or more than one job because if there is an empty slot or a slot with a non-urgent job then ALG should schedule job J in that slot instead. Because J is a non-max job there are at least two slots. We choose two slots such that at least one of them does not contain a class S job (or its part), which is possible, because any class S job is a non-max job.

Case 1: Both slots contain more jobs. Then there are at least four jobs and the lemma holds.

Case 2: One slot contains more jobs and the other slot contains an urgent job K. Then job K is not a class S job: otherwise all ALG jobs would have the same size and all the slots would have to be full. If job K is class C, P or U then it receives at most 1 vertically and nothing diagonally (because K is urgent) and the lemma holds.

Case 3: Both slots contains one urgent job. At least one of the slots does not contain class S job; hence, it contains a class C, P or U job, which receieves at most 1 and the lemma holds as in case 2. □

Lemma 7. *Let OPT job J be a scheduled red job. Then ALG job J is not a class S job.*

Proof. Suppose that ALG job J is scheduled at timeslot T and is a class S i-job. In that case timeslot T of ALG schedule is filled by i-jobs and there is one smaller job in OPT schedule at timeslot T, as these are the assumptions of case 1.2.1 in Lemma 4 that are needed to classify a job as a class S job. Therefore, for matching used for classification of jobs there are more i-jobs in ALG schedule than in OPT schedule and by the definition of matching in that case job J as only red job remains unmatched and thus cannot be classified as class S job. Contradiction. □

Theorem 1. *The competitive ratio of ALG is at most* 1.6 *in a general case and at most* 1.5 *in the tall/small case and in the non-max case.*

Proof. We described the charging scheme for black and white OPT jobs earlier. To complete the proof it remains to describe the charging scheme for red OPT jobs and to show that each ALG job receives at most 1.6 (or 1.5). According to Lemma 4 it is possible to distribute vertical edges between ALG jobs in such a way that ALG jobs can be divided to four classes C, P, U and S. Class C jobs receive at most 1/2 vertically (by definition) and at most 1 diagonally (as every job). Class M jobs receive at most 1 vertically (by definition) and nothing diagonally (by Lemma 3). Class U jobs receive at most 1 vertically (by definition) and nothing diagonally because they are urgent and therefore they cannot be scheduled later by OPT. Class S jobs receive at most 1.5 vertically and nothing diagonally (by Lemma 3). Therefore, if there is no red job, then each ALG job receives at most 1.5.

Now we describe charging scheme for red jobs. For each red (OPT) job J, there are two cases whether job J is scheduled or not by ALG. There is also a max-job in ALG schedule (from the definition of red jobs) called job K. In the first case, job J is scheduled, and it sends 1/2 vertically to ALG job K, the only ALG job in that timeslot and 1/2 to itself in ALG schedule (ALG job J). In the second case, job J is unscheduled, and it sends 0.6 vertically to ALG job K and 0.4 to the timeslot T when job J is urgent. According to Lemma 6, either there is a ALG job in timeslot T receiving at most 1 from white and black jobs which then receives 0.4 from job J, or there are some four jobs scheduled at timeslot T and they receive 0.1.

We showed that without red jobs the theorem holds. Now we show that even if we add charging of red jobs, then the upper bounds hold. Because of Lemma 5, the charging from different red jobs does not mix and we can deal with each red job independently. In the first case of the definition of charging, job K received nothing vertically from white jobs (as there is no such job in the same timeslot) and at most 1 diagonally (as every job). Therefore we can add 1/2 from the red job. ALG job J is not class S job according to Lemma 7. Therefore, it receives at most 1 vertically from white jobs (because it is in class C, P or U) and nothing diagonally (because OPT job J is red and not black) and we can add 1/2 from the red job. In the second case, the argument for job K is the same as in the first case, but we add 0.6 and therefore it receives at most 1.6 and we can add 0.1 to each of some four jobs because without charging from red jobs they receive at most 1.5, with that they receive at most 1.6. Therefore, we shown that if there are no unscheduled red jobs, each ALG job receives at most 1.5, otherwise it receives at most 1.6 and that proves the first part of the theorem.

In the first restricted case (no max-jobs) it is obvious that there are no red jobs and therefore 1.5 is the upper bound for the competitive ratio. In the second restricted case (tall/small jobs) suppose that there is an unscheduled red i-job J, which is urgent at timeslot T. Because there are no smaller jobs, ALG schedule of timeslot T must be full of urgent i-jobs. As the algorithm does not specify any preferences for choosing urgent jobs of the same size, ALG might choose job J instead of one of these i-jobs (and similarly for other unscheduled red jobs). As both jobs are urgent, such choice would not affect the remaining schedule. In that case there would be no unscheduled red jobs and therefore upper bound 1.5 holds. But even if ALG did not choose job J then there is the same number of jobs scheduled by ALG and therefore upper bound 1.5 holds. That proves the second part of the theorem. □

4 Lower Bound

Theorem 2. *The competitive ratio of every deterministic algorithm for online scheduling of parallel jobs on hypercubes is at least* 1.4.

Proof. We fix a deterministic algorithm. We consider an instance on two machines; it can be easily scaled up for more machines. The instance and the proof are summed up in Figure 3.

We start with two jobs A ($r_A = 1$, $d_A = 3$, $s_A = 1$) and B ($r_B = 1$, $d_B = 4$, $s_B = 2$). If the algorithm schedules job B in the first timeslot, then we extend the instance with an urgent job K ($r_K = 2$, $d_K = 3$, $s_K = 2$) and finish. The algorithm might schedule job A or job K, but loses the other job and it is 1.5-competitive (as OPT schedules all jobs). This is the first case in Figure 3.

Otherwise, the algorithm schedules job A in the first timeslot and OPT schedules job B. We extend the instance with an urgent job C ($r_C = 2$, $d_C = 3$, $s_C = 1$). If the algorithm schedules job B in the second timeslot, then it loses job A and it is 1.5-competitive (as OPT schedules all jobs). This is the second case in Figure 3.

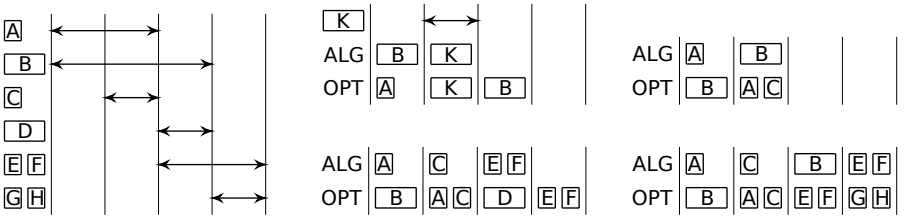


Fig. 3. The instance for the lower bound and the four cases in the proof

Otherwise, the algorithm schedules job C in the second timeslot and OPT schedules jobs A and C. We extend the instance with an urgent job D ($r_D = 3$, $d_D = 4$, $s_D = 2$) and two smaller jobs E and F ($r_E = r_F = 3$, $d_E = d_F = 5$, $s_E = s_F = 1$), If the algorithm schedules jobs E and F for the third timeslot, then it loses jobs B and D and it is 1.5-competitive (as OPT schedules all jobs). This is the third case in Figure 3.

Otherwise, the algorithm schedules job B or D in the third timeslot and OPT schedules jobs E and F. We extend the instance with two urgent jobs G and H ($r_G = r_H = 4$, $d_G = d_H = 5$, $s_G = s_H = 1$), The algorithm schedules two jobs and loses the other two jobs from jobs E, F, G, H and it is 1.4-competitive, as OPT schedules all these four jobs during last two timeslots, but loses job D. This is the fourth case in Figure 3. □

Acknowledgements. This research was partially supported by Institute for Theoretical Computer Science, Prague (project 1M0545 of MŠMT ČR) and grant IAA100190902 of GA AV ČR. Zajíček and Ebenlendr are also partially supported by Institutional Research Plan No. AV0Z10190503.

References

1. Baptiste, P., Schieber, B.: A note on scheduling tall/small multiprocessor tasks with unit processing time to minimize maximum tardiness. J. Sched. 6, 395–404 (2003)
2. Dürr, C., Hurand, M.: Finding total unimodularity in optimization problems solved by linear programs. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 315–326. Springer, Heidelberg (2006)
3. Ye, D., Zhang, G.: Maximizing the throughput of parallel jobs on hypercubes. Inform. Process. Lett. 102, 259–263 (2007)
4. Zajíček, O.: A note on scheduling parallel unit jobs on hypercubes. Int. J. on Found. Comput. Sci. 20(2), 341–349 (2009)

Verification of Causality Requirements in Java Memory Model Is Undecidable

Matko Botinčan[1], Paola Glavan[2], and Davor Runje[2]

[1] Department of Mathematics, University of Zagreb
matko.botincan@math.hr
[2] Faculty of Mechanical Engineering and Naval Architecture, University of Zagreb
{pglavan,davor.runje}@fsb.hr

Abstract. The purpose of the Java memory model is to formalize the behavior of the shared memory in multithreaded Java programs. The subtlest points of its formalization are causality requirements that serve to provide safety and security guarantees for incorrectly synchronized Java programs. In this paper, we consider the problem of verifying whether an execution of a multithreaded Java program satisfies these causality requirements and show that this problem is undecidable.

Keywords: Java memory model, multithreading, verification.

1 Introduction

The Java language specification [1] and recent work on formalization of the Java memory model (JMM) [2,3] attempt to give a precise specification of the behavior of the shared memory for multithreaded Java programs. The JMM has been designed having two goals in mind. The first one is to provide safety guarantees to programmers by:

- ensuring sequentially consistent behavior of correctly synchronized (data race free) programs, and
- promising that even for programs that are incorrectly synchronized with respect to JMM semantics (i.e., programs with data races) the values should not appear out of thin air.

The second one aims to guarantee compiler writers that common compiler optimization techniques are allowed as long as they do not violate these safety guarantees.

The original specification of the JMM was shown to have serious flaws, e.g., Theorem 1 in [2,3] does not hold, infinite executions conflict with omega ordering of default initialization actions in the definition of the JMM, it is unclear how to handle dynamic allocation in this setting, etc. Although subsequent work on JMM [4,5,6] managed to fix some of these problems, all variations of the JMM

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 62–67, 2010.

definition contain an inherent deficiency regarding decidability which we address in this paper.

The subtlest points of the JMM definition are causality requirements that serve to provide safety guarantees for incorrectly synchronized Java programs. The problem is that they are specified declaratively, and from this definition it is not evident how to effectively check them. In [7], authors deal with the problem of verifying the JMM causality requirements for a finite execution of a multithreaded Java program containing no synchronization actions, actions on final fields and external actions. They show that the problem is NP-complete, however, their result holds only under additional assumption (implicit from the proof) that all intermediate executions in the justification sequence are finite and polynomially bounded, which is generally not true for arbitrary multithreaded Java programs. In this paper, we consider what happens when this additional assumption is left out and show that the problem of verifying the JMM causality requirements for a finite execution of an arbitrary multithreaded Java program is undecidable.

The rest of the paper is structured as follows. The formal definition of the JMM is given in Section 3. Section 3 contains the main result of this paper. In Section 4, we give concluding remarks.

2 The Java Memory Model

Let us first introduce the concepts from [3,2] that are needed for understanding the definition of the Java Memory Model (JMM).

We consider a multithreaded Java program that spawns a set of threads. The execution of each thread is represented as a sequence of actions. Formally, an *action* is a tuple $\langle t, k, v, u\rangle$, where t is the thread performing the action; k is the kind of the action: read, write, volatile read, volatile write, lock, unlock, thread create, thread join or an external action; v is the variable or monitor involved in the action; and u is an arbitrary unique identifier of the action (though, for readability, we do not write u explicitly). Non-volatile read and write actions are *non-synchronization* actions, the others are *synchronization* actions. In the rest of the text, we do not deal with thread create, thread join and external actions, however, we use the notion of initialization actions for setting up initial values of shared variables.

An *execution* is a tuple $E = \langle P, A, \xrightarrow{po}, \xrightarrow{so}, W, V, \xrightarrow{swo}, \overset{hbo}{<}\rangle$,[1] where

- P is a Java program;
- A is a set of actions;
- $\xrightarrow{po}$ is the *program order* — a partial order over actions in A that is a total order over all actions preformed by the same thread;
- $\xrightarrow{so}$ is the *synchronization order* — a total order over all synchronization actions in A;

[1] Here we denote binary relations with $\xrightarrow{\alpha}$, for some label α. Transitive closure of relation $(\xrightarrow{\alpha})^+$ is denoted by $<_\alpha$, when it is a strict partial order.

- W is the *write-seen function* — a function assigning a write action $W(r)$ to each read action r in A;
- V is the *value-written function* — a function assigning a value $V(w)$ to each write action w in A;
- $\xrightarrow{swo}$ is the *synchronizes-with order* — the smallest relation over synchronization actions in A such that:
  - if a_1 is unlocking and a_2 is locking the same object, and $a_1 \xrightarrow{so} a_2$, then $a_1 \xrightarrow{swo} a_2$;
  - if a_1 is volatile writing to and a_2 is volatile reading from the same location, and $a_1 \xrightarrow{so} a_2$, then $a_1 \xrightarrow{swo} a_2$;.
- $\overset{hbo}{<}$ is the *happens-before order* — a strict partial order induced by the synchronizes-with order and the program order, i.e., $\overset{hbo}{<} = (\xrightarrow{po} \cup \xrightarrow{swo})^+$.

An execution E is *well-formed* if it obeys the Java intrathread semantics, i.e., if it satisfies the following conditions:

(1) **Each read of a variable x sees a write to x.** All reads and writes of volatile variables are volatile actions.
(2) **The synchronization order is at most an omega order**, i.e., for each synchronization action x, the set $\{y \mid y <_{so} x\}$ is finite.
(3) **Synchronization order is a strict total order consistent with program order**, i.e., $<_{po}|_{Dom(<_{so})} \subseteq <_{so}$.
(4) **Lock operations are consistent with mutual exclusion**, i.e., for all lock actions l on monitor m and all threads t (different from the thread of l) the number of locks of t before l in $<_{so}$ is the same as the number of unlocks of t before l in $<_{so}$.
(5) **The execution obeys intra-thread consistency**, i.e., for each thread t, the actions preformed by t in A are executed in the same order that would be generated if t is run as a single thread in isolation.
(6) **The execution obeys synchronization-order consistency**, i.e., for every volatile read $r \in A$, it is not the case that $r <_{so} W(r)$, and additionally, there must not exists a write w on the same variable v such that $W(r) <_{so} w <_{so} r$.
(7) **The execution obeys happens-before consistency**, i.e., for every read $r \in A$, it is not the case that $r <_{hbo} W(r)$, and additionally, there must not exist a write w on the same variable v such that $W(r) <_{hbo} w <_{hbo} r$.

A well-formed execution E is *JMM-consistent* if it satisfies the JMM causality requirements, i.e., if there exists a committing sequence of sets of actions $\emptyset = C_0 \subset C_1 \subset C_2 \subset \ldots$ such that $A = \bigcup_i C_i$, that get justified through a sequence of well-formed executions $E_1, E_2, \ldots$ of the program P. The sequences $(C_i)_i$ and $(E_i)_i$, where $E_i = \langle P, A_i, \xrightarrow{po_i}, \xrightarrow{so_i}, W_i, V_i, \xrightarrow{swo_i}, <_{hbo_i} \rangle$, are required to satisfy the following conditions:

1. $C_i \subset A_i$;
2. $<_{hbo_i}|_{C_i} = <_{hbo}|_{C_i}$;
3. $\xrightarrow{so_i}|_{C_i} = \xrightarrow{so}|_{C_i}$;

4. $V_i \mid_{C_i} = V \mid_{C_i}$;
5. $W_i \mid_{C_{i-1}} = W \mid_{C_{i-1}}$;
6. $\forall r \in A_i \backslash C_{i-1}, W_i(r) <_{hbo_i} r$;
7. $\forall r \in C_i \backslash C_{i-1}, W_i(r) \in C_{i-1}, W(r) \in C_{i-1}$.

3 Verification of the JMM Causality Requirements

We define the problem of verifying the JMM causality requirements as follows. The input to the problem is a finite well-formed execution E of a Java program P. The question we are interested in is whether E satisfies the JMM causality requirements, i.e., whether E is JMM-consistent.

Let S be an arbitrary sequential program containing no synchronization actions, no actions on final fields, no external actions, and no references to global variables. Let P be a multithreaded program with two threads T_a and T_b described as follows:

$$T_a: \quad \texttt{y} = \texttt{x}; \qquad T_b: \quad \texttt{if (y == 0) \{ S \}} \\ \texttt{x} = \texttt{1};$$

Assume that both `x` and `y` are initially set to `0` by initialization actions i_1 and i_2 in an initialization thread I, and they happen before any other action in the execution. We represent these facts by extending the program order. Let $E = \langle P, A, \xrightarrow{po}, \xrightarrow{so}, W, V, \xrightarrow{swo}, \overset{hbo}{<} \rangle$ be a finite well-formed execution of P defined by the following components:

- $A = \{i_1 = \langle I, \texttt{write}, x\rangle, i_2 = \langle I, \texttt{write}, y\rangle, a_1 = \langle T_a, \texttt{read}, x\rangle,$ $a_2 = \langle T_a, \texttt{write}, y\rangle, b_1 = \langle T_b, \texttt{read}, y\rangle, b_2 = \langle T_b, \texttt{write}, x\rangle\}$;
- $\xrightarrow{po} = \{(a_1, a_2), (b_1, b_2), (i_1, i_2), (i_1, a_1), (i_2, a_1), (i_1, b_1), (i_2, b_1)\}$;
- $\xrightarrow{so} = \emptyset$;
- $W = \{(a_1, b_2), (b_1, a_2)\}$;
- $V = \{(i_1, 0), (i_2, 0), (a_2, 1), (b_2, 1)\}$;
- $\xrightarrow{swo} = \emptyset$;
- $\overset{hbo}{<} = \{(i_1, a_1), (i_2, a_1), (i_1, b_1), (i_2, b_1), (a_1, a_2), (b_1, b_2),$ $(i_1, a_2), (i_2, a_2), (i_1, b_2), (i_2, b_2)\}$.

Then the following lemmas hold.

Lemma 1. *If S terminates, i.e., if S run as a singlethreaded program has a finite execution, then E is JMM-consistent.*

Proof. Assume that S terminates. Then the following sequence of actions $(C_i)_i$ and executions $(E_i)_i$ satisfy the JMM causality requirements for E:

E_1: - $C_1 = \{i_1, i_2\}$;
- $W_1 = \{(a_1, i_1), (b_1, i_2)\}$;
- $V_1 = \{(i_1, 0), (i_2, 0), (a_2, 0), (b_2, 1)\}$;

(note that b_2 indeed can be executed "after" b_1 since S terminates);

E_2: - $C_2 = C_1 \cup \{b_2\}$;
- $W_2 = \{(a_1, i_1), (b_1, i_2)\}$;
- $V_2 = \{(i_1, 0), (i_2, 0), (a_2, 0), (b_2, 1)\}$;

E_3: - $C_3 = C_2 \cup \{a_1\}$;
- $W_3 = \{(a_1, b_2), (b_1, i_2)\}$;
- $V_3 = \{(i_1, 0), (i_2, 0), (a_2, 0), (b_2, 1)\}$;

E_4: - $C_4 = C_3 \cup \{a_2\}$;
- $W_4 = \{(a_1, b_2), (b_1, i_2)\}$;
- $V_4 = \{(i_1, 0), (i_2, 0), (a_2, 1), (b_2, 1)\}$;

E_5: - $C_5 = C_4 \cup \{b_1\}$;
- $W_5 = \{(a_1, b_2), (b_1, a_2)\}$;
- $V_5 = \{(i_1, 0), (i_2, 0), (a_2, 1), (b_2, 1)\}$;

Lemma 2. *If E is JMM-consistent then S terminates.*

Proof. Assume that E is JMM-consistent and S does not terminate. We claim that then the action b_2 cannot be committed through any sequence of actions $(C_i)_i$, and thus cannot take place in the final execution E, implying that E is not JMM-consistent. Namely, since C_1 contains only initializations actions, it cannot contain b_2. Assume that b_2 is not contained in some C_{i-1}. This means that in the execution E_i, the read of y in T_b (the action b_1) can only see either the initial write of 0 to y (the action i_2) or the write of 0 to y performed by T_a through the action a_2. Since in both cases the condition of the if-statement is satisfied, statements of the program S get executed infinitely, thus not allowing b_2 to get executed. Therefore, b_2 cannot be committed in C_i either.

Since determining whether a sequential program S terminates is undecidable, from Lemma 1 and Lemma 2 we conclude the following:

Theorem 1. *Verification of the JMM causality requirements is undecidable.*

4 Conclusions

In this paper, we considered the problem of verifying whether a finite execution of an arbitrary multithreaded Java program satisfies the causality requirements stemming from the Java memory model. It has been shown that this problem is undecidable.

We see this result as an important weakness of the JMM specification since it shows that one cannot have a dedicated verification algorithm in the general case. One can, however, employ the JMM definition in order to develop a simple model checker that solves the problem for some specific cases ("small" with respect to the number of program instructions, threads, and especially number of data races, see [8]).

The sequential consistency memory model has also been shown to be undecidable [9]. This result, however, did not make a definite verdict on the practical aspect of the sequential consistency memory model verification [10]. Taking into account Java practitioners needs, we could also expect verifiable fragments of the JMM to appear in the future.

References

1. Gosling, J., Joy, B., Steele, G., Bracha, G.: The Java Language Specification, 3rd edn. Addison-Wesley, Reading (2005)
2. Manson, J., Pugh, W., Adve, S.V.: The Java memory model (expanded version). ACM Transactions on Programming Languages and Systems (submitted)
3. Manson, J., Pugh, W., Adve, S.V.: The Java memory model. In: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2005), pp. 378–391. ACM Press, New York (2005)
4. Cenciarelli, P., Knapp, A., Sibilio, E.: The Java memory model: Operationally, denotationally, axiomatically. In: De Nicola, R. (ed.) ESOP 2007. LNCS, vol. 4421, pp. 331–346. Springer, Heidelberg (2007)
5. Aspinall, D., Sevcik, J.: Formalising Java's data-race-free guarantee. In: Schneider, K., Brandt, J. (eds.) TPHOLs 2007. LNCS, vol. 4732, pp. 22–37. Springer, Heidelberg (2007)
6. Aspinall, D., Sevcik, J.: Java memory model examples: Good, bad and ugly. In: Proceedings of the 1st International Workshop on Verification and Analysis of Multi-threaded Java-like Programs, VAMP 2007 (2007)
7. Polyakov, S., Schuster, A.: Verification of the Java causality requirements. In: Ur, S., Bin, E., Wolfsthal, Y. (eds.) HVC 2005. LNCS, vol. 3875, pp. 224–246. Springer, Heidelberg (2006)
8. Manson, J.: The Java memory model. PhD thesis, University of Maryland, College Park (2004)
9. Alur, R., McMillan, K.L., Peled, D.: Model-checking of correctness conditions for concurrent objects. Inf. Comput. 160(1-2), 167–188 (2000)
10. Sezgin, A., Gopalakrishnan, G.: On the decidability of shared memory consistency verification. In: Proceedings of the 3rd ACM & IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE 2005), pp. 199–208. IEEE, Los Alamitos (2005)

A Team Object for CoArray Fortran

Robert W. Numrich

Minnesota Supercomputing Institute
University of Minnesota, Minneapolis, MN

Abstract. This paper outlines the features of a team object for CoArray Fortran to support multi-disciplinary applications. It combines object-oriented design, supported in Fortran 2003, with the parallel coarray model, supported in Fortran 2008. It extends the coarray model by adding state to a coarray object. The compiler and run-time environment use this state to dereference co-indices relative to the team that created the object. Methods are associated with a team object for synchronization, memory allocation and collective operations across the team.

1 Introduction and Motivation

Fortran is now a true object-oriented language as defined by the Fortran 2003 standard [3]. The next standard, currently called Fortran 2008, will likely include the parallel coarray extension as a standard feature [4,5,6]. Although earlier versions of the coarray model included a primitive idea of teams, that is, a subset of images that work together on a particular problem, the team concept was considered insufficiently well defined and was deferred from the Fortran 2008 standard until a later revision. Since the team concept is open for discussion, this paper proposes a new way to think about teams within the coarray model. It is only a proposal, and there should be no assumption that it will or will not become part of the standard language. Mellor-Crummey and co-workers [2] have recently proposed an alternative approach for defining teams. The overlap between the two proposals is large, but they are not identical.

The motivation for this proposal is to design teams in such a way that two stand-alone codes, written in the coarray model, can be coupled together with only small changes to the original codes. In other words, the components in a coupled code look pretty much the same as they did as stand-alone codes. The design is to some extent similar to a communicator as defined by the Message-Passing Interface (MPI) [1,7]. The two ideas are not identical. The team object is designed to fit the coarray model with as few new features added to the existing model as possible.

The main problem that needs to be solved is related to memory allocation of coarrays and dereferencing of codimensions. In the current coarray model [5], an allocatable coarray,

```
real, allocatable :: x[:,:]
```
(1)

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 68–73, 2010.

with multiple codimensions is allocated across all images,

```
allocate(x[p,*])
```
(2)

The co-indices `[q,r]` representing a coarray on a remote image,

```
y = x[q,r]
```
(3)

are dereferenced relative to all the images based on the codimensions in the `allocate` statement. Making teams useful requires allocation to occur across just members of a team and dereferencing of co-indices to occur relative to the members of the team. Similarly, synchronization should be allowed across a team rather than across all images and collective operations among team members alone need to be defined. The team object defined in this paper is one way to extend the coarray model to provide these features.

2 Class AbstractTeam

A team object is an extension of an `AbstractTeam` object defined, for example, in a module named `ClassAbstractTeam`,

```
module ClassAbstractTeam
  Type,public,abstract :: AbstractTeam
  contains
   procedure(myTeam),public,pass(this),deferred    :: isMyTeam
   procedure(getSize),public,pass(this),deferred   :: getTeamSize
   procedure(getIndex),public,pass(this),deferred  :: getTeamIndex
   procedure(getList),public,pass(this),deferred   :: getTeamList
   procedure(alloc),public,pass(this),deferred     :: allocate
   procedure(dealloc),public,pass(this),deferred   :: deallocate
   procedure(sync),public,pass(this),deferred      :: sync
   procedure(sum),public,pass(this),deferred       :: sum
   procedure(max),public,pass(this),deferred       :: max
   procedure(min),public,pass(this),deferred       :: min
   procedure(run),public,pass(this),deferred       :: run
  end Type AbstractTeam
  abstract interface
   logical function isMyTeam(this)
   import :: AbstractTeam
   Class(AbstractTeam),intent(in) :: this
  end interface
end module ClassAbstractTeam
```
(4)

The reader unfamiliar with Fortran 2003 should consult a reference book to decipher this code sample [3].

A concrete class that extends this abstract class must provide the deferred procedures that match specified interfaces such as the one shown for the function

`isMyTeam()`. This function returns a logical true if the image that invokes the function is in the team and logical false if it is not in the team. The function `getTeamIndex()` returns the invoking image's index relative to the team, and the function `getTeamSize()` returns the number of images in the team. The function `getTeamList()` returns a list of images in the team. The interfaces for these functions are omitted for lack of space.

Some of the deferred procedures listed for the abstract class imply a change in the coarray model. Each team, for example, defines its own memory allocation, deallocation and synchronization procedures. In the current model, allocation of a coarray is a collective operation across all images. In the team model, allocation takes place only across members of the team. The compiler, then, for each coarray, must keep track of its team context, and it must dereference codimensions relative to the team. Synchronization takes place only among team members, and collective operations, such as `sum, max, min`, take place only among team members.

The programmer assigns work to each team by providing a `run` procedure specific to the team perhaps something like example (10) shown in section 4. This procedure might, for example, be a wrapper around an ocean model for one team and a wrapper around an atmosphere model for another team. Inside one of these wrappers, the programmer allocates coarrays and uses synchronization procedures, as if it were a stand-alone code written in the coarray model, but uses the procedures associated with the particular team that invoked the `run` procedure.

Each concrete team supplies a constructor, as a function with the same name as the team class, that returns a team object of that type. For example, for a team of type `AllTeam` that includes all the images, the code sample,

```
Type(AllTeam) :: all
real,allocatable :: x[:]
real :: s
  all = AllTeam()
  call all%sync()
  call all%allocate(x[*])
  s=all%sum(x[*])                                    (5)
```

creates a team named `all` and uses the `all%sync`, `all%allocate` and `all%sum` procedures associated with that team. This class essentially represents the existing coarray model. All images must invoke the constructor `AllTeam`, so they all know they belong to the team, and the procedures `all%sync`, `all%allocate`, `all%sum` and so forth are collective procedures across all images.

The programmer creates other kinds of teams by invoking a team constructor specific to each team class. These team classes can be created by the programmer, as extensions of the abstract team class or through inheritance from other concrete classes, and the programmer must provide a constructor for the class along with all the other deferred procedures. The programmer may, of course, add other procedures and data components specific to a particular class.

For a coarray allocated by a team, the compiler must dereference codimensions relative to the team. Keeping track of team information can be done, for example, by adding a pointer to the team in the allocated object's dope vector. This extra layer of indirection may degrade performance, and it is important to define a team in such a way to reduce this overhead. For example, a simple team might be one that includes a contiguous subset of images.

```
Type(ContiguousTeam) :: air, ocean
  p=num_images()
  air = ContiguousTeam(1,p/2)
  ocean = ContiguousTeam(p/2+1,p)
  if(air%isMyTeam()) then
     call air%run()
  elseif(ocean%isMyTeam()) then
     call ocean%run()
  end if
```
(6)

The constructor for a contiguous team accepts two arguments specifying the beginning and end of the image numbers contained in the team. The compiler then uses a simple offset to dereference codimensions.

3 Collectives

Within a team, the programmer may want to form a subteam to compute, for example, a global reduction across a particular codimension. A `CoDimTeam` class might provide this capability. Its constructor might accept three arguments specifying the row relative to the codimension of a specific coarray, a stride between images in the team, and the number of images in the row.

```
Type(CoDimTeam) :: rowTeam
real :: x[q,*]
real :: s
  rowTeam=CoDimTeam(p,q,r)
  s = rowTeam%sum(x)
```
(7)

Each object of class team must provide a set of collective procedures. These procedures should be written to optimize communication based on the known relationships among members of the team.

4 Communication among Teams

Data exchange among different teams can take place through buffers created across all images and passed to various subteams. Within each subteam, coindices for this buffer are dereferenced relative to the all the images because it was allocated across all images. Two different teams can read data from or write data to this buffer using coarray syntax. The programmer is responsible for keeping track of how image numbers for each team are related.

```
Type(AllTeam) :: all
Type(ContiguousTeam) :: air,ocean
real,allocatable :: buffer(:)[:]
  call all%allocate(buffer(n)[*])
  if(air%isMyTeam) call air%run(buffer)
  if(ocean%isMyTeam) call ocean%run(buffer)          (8)
```

The programmer is responsible for maintaining memory consistency using proper synchronization procedures. A team may synchronize with itself or with another team using overloaded versions of the synchronization procedure, for example, as shown in the code sample,

```
if(air%isMyTeam()) then
  call air%sync()
  call air%sync(ocean)
elseif (ocean%isMyTeam()) then
  call ocean%sync()
  call ocean%sync(air)
endif                                                (9)
```

Synchronization between pairs of teams, of course, must happen in pairs and the programmer must guard against deadlocks.

As an example of two teams interacting with each other, consider the `run` procedure provided by the programmer for the ocean team.

```
subroutine run(ocean,air,buffer)
  Type(ContiguousTeam),intent(in) :: air,ocean
  real,intent(inout) :: buffer(1:)[*]
  integer :: airImage
  !------local arrays-----
    airImage = air%getTeamList(1)
    call ocean%sync(air)
    airBoundayConditions(:) = buffer(:)[airImage]
    !-----some ocean work------
    call ocean%sync(air)
    airBoundayConditions(:) = buffer(:)[airImage]
    !-----some more ocean work------
end subroutine run                                   (10)
```

The atmosphere team writes boundary conditions needed by the ocean team into the coarray buffer on the first image in its team. The ocean team does the same with boundary conditions needed by the atmosphere into the buffer on its first image. The programmer obtains the global image index for the first member of the air team using the function `air%getTeamList(1)` and uses it as a co-index to point to the buffer containing the boundary conditions supplied by the atmosphere team. Since the buffer coarray was allocated by team `all`, the compiler dereferences co-indices relative to all images.

5 Remarks

This proposal adds something new to the coarray model. Codimensions have always been interpreted locally, within the scope of each procedure, with the hidden assumption that the compiler interprets a co-index relative to all images fixed at run-time. In this new proposal, each coarray carries with it a team state. The compiler still interprets codimensions locally but within the context of the team that allocated the coarray. A team object contains information, for each concrete instantiation, that the compiler uses to interpret codimensions and to dereference co-indices.

This extra overhead may lead to a degradation in performance. If the images in a team are some random collection, with no obvious rule that relates one to another, the compiler may need to generate run-time code to compute an image number from a co-index. But simple teams, like contiguous teams or codimension teams, can be defined in such a way that the compiler can use a simple formula to dereference co-indices. Nonetheless, within this new model, the programmer, by extending the abstract team, may define a team based on a general graph that defines a complicated relationship among the members of the team. The programmer must be willing, then, to accept the extra overhead.

This proposal remains within the SPMD model assumed by the coarray model. A team is a subset of the fixed number of images set at run-time. It resembles an MPI intra-communicator within an MPI group. But the model could be extended to add a new dynamic feature to the model where the `run()` procedure might become, say, a `spawn()` procedure that launches a new set of images for each team. This feature would resemble an MPI inter-communicator between different MPI groups. Adding this dynamic feature to the coarray model would involve extensive run-time support in addition to compiler support.

References

1. Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W., Snir, M.: MPI: The Complete Reference. The MPI-2 Extensions, vol. 2. MIT Press, Cambridge (1998)
2. Mellor-Crummey, J., Adhianto, L., Scherer III, W.: A New Vision for Coarray Fortran. In: Proceedings PGAS 2009, October 5-8. George Washington University (2009)
3. Metcalf, M., Reid, J., Cohen, M.: Fortran 95/2003 Explained. Oxford University Press, Oxford (2004)
4. Numrich, R.W., Reid, J.K.: Co-arrays in the next Fortran standard. ACM Fortran Forum (2005)
5. Reid, J.: Coarrays in the next Fortran Standard. ISO/IEC JTC1/SC22/WG5 N1787 (2009)
6. Reid, J., Numrich, R.W.: Co-arrays in the next Fortran Standard. Scientific Programming 15(1), 9–26 (2007)
7. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: The Complete Reference, 2nd edn., vol. 1. MIT Press, Cambridge (1998)

On the Definition of Service Abstractions for Parallel Computing

Hervé Paulino

Departamento de Informática, Universidade Nova de Lisboa, Portugal
herve@di.fct.unl.pt

Abstract. The availability of real parallelism in multi-core based architectures has resurrected the interest in concurrent computing in general, and parallel computing in particular. New languages and libraries have been recently proposed to increase productivity in the context of these architectures. In this paper we present a novel approach that resorts to the service abstraction for annotating parallelism.

1 Introduction and Motivation

Multi-core CPU architectures are ushering a new era in computer design and organization. Recent years have seen parallelism become the driver for CPU performance increase, making multi-core CPUs the de-facto standard in modern CPU design. In fact, IBM researchers envision high-bandwidth interconnected nodes composed of several multi-core processors with non-uniform memory hierarchies as the emerging processor organization paradigm [1].

The widespread infrastructural support for real parallelism has resurrected the interest in concurrent computing as a whole and in parallel computing in particular. MPI [2] and OpenMP [3] have been, until now, the community standards for, respectively, the message-passing and shared-memory paradigms. However, both have generally accepted limitations. The MPI programming model is error-prone, forces per-processor implementation of the algorithms [4] and does not have implementations that support the deploying of applications in heterogeneous environments. OpenMP features a higher degree of abstraction, usually only requiring minor modifications on the sequential algorithms, but the global shared-memory model limitation is inadequate for these new architectures.

This state-of-the-art has motivated the definition of new programming abstractions and runtime systems to address the development and consequent deployment of applications in these new generation of architectures. DARPA's (Defense Advanced Research Projects Agency) HPCS (High Productivity Computing Systems) program funded research to tackle these issues, giving birth to the X10 [1], the Chapel [4], and the Fortress [5] programming languages. Intel is also researching on the increase of productivity in multi-core architectures, proposing the Ct [6], and Intel Threading Building Blocks (TBB) [7] libraries.

This paper introduces initial research on a novel approach based on services for the programming of parallel applications. Services are nowadays an established

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 74–81, 2010.

programming abstraction in the development of loosely-coupled distributed applications. It is our purpose to apply its modularity and intuitiveness to parallel computing and, with that, provide a good framework for the design of non-sequential code.

We intend to decouple functionality from resource awareness and provide a two-level application construction scheme. On a first level functionalities are aggregated into services that are implemented and accessed as if they where a single entity in a global addressing space. The actual number of service instances and their mapping into the processor pool is defined at a second level.

The remainder of the paper is structured as follows: section 2 presents our service-based model; section 3, its relation to the current state-of-the-art; and, finally, section 4, draws some conclusions and guidelines for future work.

2 A Service-Based Approach to Parallel Computing

We begin this section by giving a more comprehensive explanation of what is a service in our model. A service is an abstraction that gathers (normally) related functionalities and provides them to the client as an interface. All the services required by an application must be defined within, and compiled with the remainder of the application. We are not in the presence of a distributed computing model. There is no loose-coupling, no need for registering and searching for service providers, nor for interoperability at this level.

A program is a collection of services plus a *main* section, that bootstraps the execution. Although it is not explicit in the definition of a service, the later may have several execution flows running in distinct computing elements, such as distinct CPUs or computers. The number of execution flows and their mapping in the execution environment is only defined at deployment-time.

Figure 1 illustrates the deployment of an application in a given architecture. On the top left corner we have the application, which defines and uses services S_1, S_2, and S_3. Below, we have its mapping in the execution environment. This information defines the number of instances of each service and how these map in the processor pool of the target architecture. The syntax is declarative and allows for statements of the following three forms:

```
n of S              // Launch n instances of service S
n of S in a         // Launch each of the n instances of S in a architecture of type a
n of S groupby a    // Group the n instances of S in nodes of type a
```

where n denotes the number of instances, S the service identifier and a the target architecture, for instance, dual, quad and node (for multiprocessor nodes).

Both the application and the mapping are fed to the application launcher that has the responsibility of ensuring the matching between the later and the target architecture. In figure 1 this architecture is composed of a set of single, dual and quad-core machines. Note that the *main* section will execute in the machine where the application is launched.

Next, we present our service-based programming model for the writing of parallel applications.

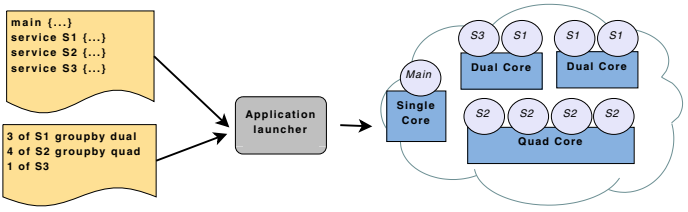


Fig. 1. Launching an application

2.1 The Programming Model

The model is orthogonal to the common general-purpose sequential and concurrent programming languages. Thus, it does not define a Turing-complete model, but rather a set of primitives to annotate parallelism in applications. Throughout this subsection we will use the C programming language to illustrate our concepts. We are not defining a concrete syntax but rather making an exercise to present the abstractions of the model.

Defining and Using Services: Consider the next three mathematical functions: square-root (textsfsqrt), power (pow), and the Fibonacci function (fib) written in the C language.

```
double sqrt (double d) { ... }
double pow(double b, long e) { ... }
long fib(long n) { ... }
```

Our objective is to have simple and intuitive abstractions, and with that increase productivity. The syntax for the aggregation of these functions in a service can be as simple as:

```
service Math {
   double sqrt (double d) { ... }
   double pow(double base, long exp) { ... }
   long fib(long n) { ... }
}
```

For the sake of modularity a service may define local operations for internal use only. In Java this can be easily achieved with the notion of private method. In other languages, such as C, a special keyword (**local**) may be required. An example that adds logging capability to the Math service follows:

```
service Math {
   ...
local:
   void log(char message[]) { ... }
}
```

Service operation invocation: The invocation of service operations is synchronous, meaning that the execution flow of the caller blocks until the result is available. The syntax is borrowed from method selection in object-oriented languages, as illustrated in the recursive implementation of the fib operation below.

```
   long fib(long n) { return n < 2 ? 1 : Math.fib(n-1) + Math.fib(n-2); }
```

Concurrent invocations: In order to have full concurrency, both in the server as in the client, we define a syntax for concurrent service operation invocation. Both the left and right-hand sides of an attribution become sequences. The left-hand side must hold the variables target of the operations' results, whilst the right-hand side must hold the actual invocations. The mapping is given by the order of the elements in the sequences. Note that synchronous nature of the operations is kept, the execution flow only proceeds to the next instruction when all the results have been received. With this new feature we can provide a new implementation of the fib operation where its invocations are done concurrently.

```
long fib(long n) {
   if (n < 2) return 1;
   long x, y;
   x, y = Math.fib(n-1), Math.fib(n-2);
   return x+y;
}
```

This particular implementation raises a question: are the recursive invocations of fib performed locally (by the current instance) or remotely (by some other instance, if such instance exists)? Since invocations are synchronous, the local processor will become idle after performing the calls, and can thus be used to perform other computations. In this context, a natural choice would be to execute at least one of the operations locally.

We provide this control in a simple way, the qualification of the operation with the service's identifier denotes a remote execution, while the non-qualification denotes local execution. Thus, to execute fib(n-1) locally we only need to change the statement to: x, y = fib(n-1), Math.fib(n-2).

Asynchronous invocations: Asynchronous invocations of service operations is also featured in the model. The **async** keyword allows the definition of an handler (a local function in C) that is triggered when the result arrives, and has the purpose of handling such result.

An handler may be parameterized, being that the last parameter must stand for the incoming result. As we be clear in the two examples below, this parameter is transparent to the handler setting. The examples focus on the asynchronous invocation of the Math.fib operation. The first simply prints the received result, while the second assigns it to a variable passed by reference.

```
async Math.fib(n) : printRes();
```

```
void printRes(int res) {
  printf ("%d", res); }
```

```
int x;
async Math.fib(n) : assignRes(&x);
```

```
void assignRes(int *var, int res) {
  *var = res; }
```

The invocation also requires a dedicated syntax: S.addArray(a#[10]) stands for sending the first 10 elements of array a, which can have a far superior length.

Sharing data among service instances: The Math example we introduced in the beginning of this subsection is an example of a stateless service. However, stateful services can also be implemented by defining a set of variables global to

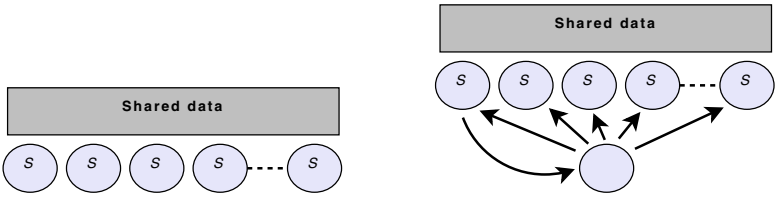


Fig. 2. (a) State sharing, (b) Shared parameters and returning shared data

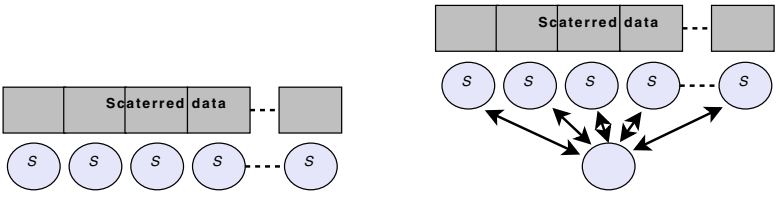


Fig. 3. (a) State scattering, (b) Scattered parameters and returning scattered data

the service. For instance, an operation may allow for the uploading of data to a service for its posterior manipulation by invoking one or more operations.

As stated before, although it is not explicit in the language, a service may be executed by multiple flows of execution. As illustrated in figure 2 (a), these may share part of the service's state. In fact, each service instance holds a local copy of the shared data. This allows read operations to be local, while writes require inter-instance communication and synchronization.

The sharing of data can also be done at parameter level, which means that a value passed to the operation must be sent to all instances processing that invocation (figure 2 (b)). As is also illustrated in the figure, the return of a shared variable must be performed by only one the instances, to avoid the reception of multiple copies.

The next example illustrates a case where all instances of service DataMining share variable data and hold a private copy of variable max. In turn, operation process manipulates a shared array (input) received as argument. Shared variables must be qualified with the **shared** keyword.

```
service DataMining {
  char data[] : shared;
  int max = 1024;
  void upload(char d[length]) { data = d; }
  void process(char input[length] : shared) { ... }
}
```

Data parallelism: In order to achieve data parallelism and improve performance, non-scalar values can be scattered among the instances of a service. As with shared values, this can be applied to both state (figure 3 (a)) or parameter variables (figure 3 (b)). In the later case, each instance receives only a subset of the entire value. The return of a scattered value requires the collection and integration of all its subsets.

An classic and intuitive example of data parallelism is the matrix multiplication problem. Each element of the result matrix is computed from one row of the first matrix (M1) and one column of the second (M2). Thus, we can partition both matrices in such way that each instance only receives only a subset of the rows of M1 and of the columns of M2. In our model, scattered arrays must qualify the dimension to scatter with the **scattered** keyword, as is coded in the following matrix multiplication implementation:

```
char[][] mul(char m1[nRows : scattered][nCRs],          // scattered by rows
             char m2[nCRs][nColumns : scattered] ) {   // scattered by columns
  char **m = calloc(nRows*nColumns, sizeof(int));
  for (int i=0; i < nRows; i++)
    for (int j=0; j < nColumns; j++)
      for (int k=0; k < nCRs; k++) m[i][j] += m1[i][k]*m2[k][j];
  }
  return m;
}
```

nRows, ncolumns and nCRs denote the length of the respective arrays dimensions. Note that later appears twice in the function's signature, which requires a static analysis to ensure value unification.

Data partitioning is, however, not always as linear as in the matrix multiplication example. Consider the problem of checking if one string is a substring of another. A linear partition will not cover the case where the string to match crosses the boundaries of the partition.

To provide a modular framework that can coupe with problem-specific partitions, we introduce the concept of distribution that can also be found in the Fortress language [5]. A distribution is the set of ranges of the target array to be supplied to a given service instance. To ease the implementation of these distributions, we defined a set-based syntax that allows for operations over ranges.

A distribution is always parameterized by the index of the instance to which the range is to sent, the number of instances target of the distribution, and the length of the array to be scattered. Other parameters can be added, as in the next example that implements a distribution for the substring search example. The added parameter denotes the length of the substring to find.

```
dist myDist(int subStringLen, int instance, int nInstances, int arrayLen) {
  int len = arrayLen/nInstances + subStringLen - 1;
  return [instance*len, min(instance*len+len, arrayLen-1)];
}
```

```
int findSubStr(char str[sLen] : myDist(sLen), char subStr[ssLen]) { ... }
```

Scattered but locally shared variables: The mapping of services into processors that share a memory hierarchy may sometimes profit from the sharing of the scattered data within the node (figure 4). Only the services grouped by the **groupby** annotation in the mapping stage can make use of this feature.

Merge (or reduce) data: When multiple service instances are computing an operation in parallel it is often necessary to merge, or reduce, the multiple received results to compute the final one. Our approach is to associate a merging function to the variable holding to result to return. The next example merges

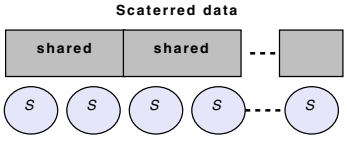


Fig. 4. Scattered but locally shared variables

the existing copy of a local array (x) with a new incoming result, computing, for each cell, the maximum of the values of that same cell in both arrays. Naturally some of the most used merging operations can be provided in a library.

```
service SomeService {
  int[] someOperation() {
    int x[] : mergewith f;
    ...
    return x;
  }
local:
  void f(int new[len]) {
     while(--len) if (new[len] > x[len]) x[len] = new[len];
}}
```

Process synchronization: As in many other languages, a keyword (**atomic**) is defined to delimit atomic blocks. Its unbound use (**atomic** {...}) ensures exclusive access to all the shared variables in the block, as if a lock upon all is performed before entering the block. Deadlock-free algorithms must be implemented. Its binding to a variable of type **sync** ensures that the delimited sequence of operations executes atomically, even if it does not access shared variables. An example follows:

```
sync s;
atomic s { printf ("Hello");  printf (" world!\n"); }
```

Memory barriers are implicitly placed whenever an access is made to a shared variable. We are yet to define if explicit barriers are to be included.

3 Related Work

The objective of our work is closely related to DARPA's HPCS funded languages and Intel's multi-core related research. Due to space restrictions we chose to focus solely on the first set, since the second relates only to shared-memory models. IBM's X10 [1] is language specially designed for NUCC architectures. As Cray's Chapel [4], it partitions the global addressing space into localities, enabling affinities between processes and processors. An approach quite far from our service abstractions. Even further is Sun's Fortress [5] that provides a syntax close to mathematical notation. Computation is parallel by nature, thus, instead of providing constructs for parallel loops (as X10 and Chapel) it provides means to serialize them. Distribution imposes parallel structures on generators, the abstractions that define how data-structures map into the target architecture.

All three languages support the asynchronous spawning of tasks, much like in Cilk [8] and pSystem [9] and array sub-languages to handle array specifics, such as

sparse matrices. Synchronization is achieved through the usual atomic code blocks. Although the use of localities and array sub-languages provide a higher degree of flexibility and abstraction, we think that the programming model is still too close to what is done in Cilk [8], pSystem [9], or UPC [10]. Fortress' idea of parallelism by default seems to be a good path to follow. Nonetheless, regarding distributed memory, we feel that the service approach is more intuitive and modular.

4 Conclusions and Future Work

The main objective of this paper is to expose our approach to the problem of increasing productivity in architectures with real parallelism. It is our opinion that the use of an established and intuitive abstraction, such is *the service*, provides a good framework for designing of non-sequential code.

The model here proposed is still at an early development stage. Work in progress focuses a formal definition and prototype implementations. A formal framework will allow us to obtain correctness properties, such as the lack of deadlocks, while prototype implementations will allow us to attest the behavior of the model in large scale applications, both performance and code maintenance. Currently implementations in Java and C are planed.

References

1. Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., von Praun, C., Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing. SIGPLAN Not. 40(10), 519–538 (2005)
2. Message Passing Forum: MPI: A Message-Passing Interface Standard. Technical Report UT-CS-94-230, University of Tennessee, Knoxville, TN, USA (May 1994)
3. OpenMP Architecture Review Board: OpenMP Application Program Interface v 3.0 (May 2008)
4. Callahan, D., Chamberlain, B.L., Zima, H.P.: The cascade high productivity language. In: Ninth International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004), pp. 52–60 (2004)
5. Allen, E., Chase, D., Hallett, J., Luchangco, V., Maessen, J.W., Ryu, S.: Jr., G.L.S., Tobin-Hochstadt, S.: The Fortress Language Specification. Technical report, Sun Microsystems, Inc. (2007)
6. Gholoum, A., Sprangle, E., Fang, J., Wu, G., Zhou, X.: Ct: A Flexible Parallel Programming Model for Tera-scale Architectures. Intel Whitepaper (October 2007)
7. Reinders, J.: Intel Threading Building Blocks. O'Reilly & Associates, Inc., Sebastopol (2007)
8. Blumofe, R.D., Joerg, C.F., Kuszmaul, B.C., Leiserson, C.E., Randall, K.H., Zhou, Y.: Cilk: An Efficient Multithreaded Runtime System. In: Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), pp. 207–216 (July 1995)
9. Silva, F., Paulino, H., Lopes, L.: di_pSystem: A Parallel Programming System for Distributed Memory Architectures. In: Proceedings of the 6th European PVM/MPI Users' Group Conference, pp. 525–532. Springer, Heidelberg (1999)
10. El-Ghazawi, T., Smith, L.: UPC: uUnified Parallel C. In: SC 2006: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, p. 27. ACM Press, New York (2006)

Performance Debugging of Parallel Compression on Multicore Machines

Janusz Borkowski

Polish-Japanese Institute of Information Technology,
86 Koszykowa Str., 02-008 Warsaw, Poland
janb@pjwstk.edu.pl
Infobright, www.infobright.com

Abstract. The power of contemporary processors is based more and more on multicore architectures. This kind of power is accessible only to parallel applications, which are able to provide work for each core. Creating a scalable parallel/multithreaded application efficiently using available cores is a difficult task, especially if I/O performance must be considered as well. We consider a multithreaded database loader with a compressing function. The performance of the loader is examined from a number of perspectives. Because compression is a computationally intensive task, parallel execution can potentially provide a big advantage in this case. A list of performance related areas we encountered is presented and discussed. We identify and verify tools allowing us to deal with specific performance areas. We find out, that only an orchestrated employment of several tools can bring the desired effect. The discussion provides a general procedure one can follow when improving the performance of multithreaded programs. Key performance areas specific to the database loader are pointed out. A special interest is directed towards performance variations observed when many parallel threads are active on a multicore CPU. A significant slowdown of computations is observed if many threads are computing simultaneously. The slowdown is related mainly to memory access and cache behavior and it is much larger for Core2 Quad system than a dual Xeon machine.

1 Introduction

The increase of processor power has been traditionally bound to the increase of clocking frequencies. However, new CPU architectures achieve their high processing capacity differently. A single modern CPU contains a few computing cores [1,2]. The cores can work in parallel, thus delivering higher performance at a lower cost, than a single core working at a very high clock rate. Facing this trend, scientists and engineers have adopted multicore CPUs in newly designed computing clusters and servers e.g. [3]. The efficiency of multicore processors has become a separate direction of research, aimed at performance evaluation and finding better architectural solutions [4]. This research has been concentrated mainly around High Performance Computing applications. On the other hand,

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 82–91, 2010.

multicore processors were originally designed to deliver more processing power at a lower cost and with lower electric power consumption to business enterprises.

In the world of business applications, databases play a leading role. Standard databases can benefit from multicore CPUs by allowing for more concurrent clients, distributing clients' queries among available cores (inter-query parallelism), e.g. MySQL, High-end database systems can execute a single query in parallel (intra-query parallelism). In both cases, the possible gain due to the usage of multicore CPUs should be investigated in separation from HPC application, due to different program characteristics. While HPC programs are computationally intensive, database engines deal with accessing large amounts of data – they tend to be more I/O intensive. In-memory databases are an exception here, but we will concentrate on main stream disk-based databases. While it is relatively easy to take advantage of a multicore CPU by assigning a separate core for each database client, more can be gained if a client (a single query) can use all available processing power. An interesting proposal for employing multicore processors in databases is presented in [5]. The authors claim, that as the number of cores per CPU grows, soon a system will not be able to use all the cores efficiently because of memory access bottleneck. Therefore some of the cores should only prefetch data to cache for the convenience of other cores. Our results shown later in this paper let us agree only partially with this idea. The impact of cache performance in database systems becomes more important as servers use more and more memory and are equipped with CPUs coupled with large caches. In [6] the performance of database systems on multicore chips is analyzed in respect to parameters of CPU cache. While this work presents an excellent overview of cache-related problems and provides important suggestions, it deals with database engines executing standard database workload. We concentrate on the load process, which is separated from the normal database operation and has different characteristics. Data warehouses can be defined casually as databases storing very large amounts of data to be analyzed. The data is loaded into a data warehouse and is not (or rarely) modified later. Some of the specific features of a data warehouse are as follows:

1. Stored data sets can be very large (many TB)
2. Data are accessed mainly for reading
3. The data is loaded usually periodically, e.g. once a day
4. The data load process should be quick

The last point is important, because it is often necessary to load many GB of data in a few hours e.g. during a night break. Therefore the load speed is an important factor in data warehouse product market. Quickly growing processing capacity of processors made it possible to shift part of the strain from disk to CPUs in data warehouse processing. Data compression makes data smaller, so disk usage becomes less intensive, but a CPU must compress data during load. This tradeoff can pay off. Therefore some database systems start using data compression techniques nowadays [7,8].

In this paper we describe a data loader – a program which reads source data files and stores compressed data in database tables. The loader is a part of the

Infobright database [9]. The need for data load speed has driven us to develop a multithreaded version of the loader. Because the application is partially irregular and the inter-thread dependencies are sophisticated, we could not employ standard parallel programming frameworks like Open-MP. We used pthreads instead. The performance of the obtained solution and methods of investigating it are the main topics of the paper. The next section describes performance problems, which can be encountered in a multithreaded parallel application with heavy I/O. Suitable performance monitoring tools are introduced there. Section 3 explains the functionality of the loader and presents performance problems we found in it. Performance profiling results are shown and the roots of the problems are explained. Conclusions are communicated in the last section.

2 Different Tools for Different Performance Problems

The performance of an I/O intensive multithreaded application, like our data loader, is much more difficult to investigate than the case of a simple computational process. A number of factors can be collectively responsible for performance deficiencies.

1. *Computationally intensive code* can slow down the overall progress. The code can use an inefficient algorithm or can be implemented inefficiently.
2. *Stalls caused by disk access* can exist. The program can wait for data read from disk and it can wait until data are written to the disk.
3. *The parallelization method* can be not efficient. If not enough work is extracted for parallel threads, then the hardware capabilities are not fully utilized.
4. *The coordination between threads* can cause a thread to wait one for another, thus serializing computations.
5. *Access to shared data* guarded by mutexes can create bottlenecks.
6. *Thread interaction* can have a negative impact on the performance. Memory and disk bandwidth can be exceeded if threads use them simultaneously. A thread can remove from the cache data used by another thread.
7. *Memory allocations* done by all running threads can cause swapping. Swapping slows done program execution by orders of magnitude.

Unfortunately, there is no single tool which can take all the performance factors into account. It is necessary to use specific tools for specific problems separately.

Program execution profiling is a common method for finding hot-spots within the program code. There two kinds of profiling: a) sampling b) tracing. Sampling checks periodically the location of the execution point. A good tool is OProfile [10]. It can do sampling with a low overhead using normal program binaries, giving statistics derived from CPU hardware counters, e.g. about the percentage of time spent in various methods. Unfortunately no stalls caused by I/O or mutexes are taken into account. Also the coordination between threads and parallelization method are not made visible.

Valgrind simulates program execution and it can give very detailed information about time spent in each function, but it does not simulate multithreaded programs properly for performance considerations [11]. Therefore we will not consider it any more.

Specialized tools can be used to instrument the code to measure the actual time spend in given functions. Such tools exist for profiling standard libraries e.g. for MPI. We have developed our own simple tracing code, called FET (Function Execution Times). The code, inserted into key methods, measured the time a method was active – its stack frame existed. The time measured in this way included all kind of stalls, which could elongate the actual method execution time.

It turned out to be difficult to find tools accurately measuring I/O waiting time. To get an estimate of it we used system monitor available in Gnome desktop and vmstat system command. We applied a comparative approach as well, checking the difference between the elapsed loading times once when the data had to be read physically from the disk and next when the data was already present in the system disk cache memory.

To our knowledge, the best existing tool, specifically targeted towards thread performance monitoring, is Intel Thread Profiler (do not confuse it with Thread Checker or Sun Studio Thread Analyzer, which are specialized in detecting race and deadlock conditions). It is a commercial tool measuring and displaying graphically different aspects of threaded program execution. Unfortunately, this tool was not available for us and additionally this tool is officially available for Windows only, while our software runs mainly on Linux. We were aware of another problem also. Our loader creates a lot of short living threads, making it difficult to monitor them in a sensible way using such a tool. We would get a rather incomprehensible for reading picture with thousands of life-lines. Sun Studio contains also some useful profiling tools, however we couldn't find there the functionality offered by Intel Thread Profiler for C/C++ programs.

Lacking the commercial tools, we developed simple in-house solutions. We measured the thread life time using our FET code mentioned above, taking a summarized time across threads doing the same kind of job (see explanation in the next section). This way we got a good insight into the complexity of particular jobs. To check the time spent on waiting for a particular mutex, we created another simple lock-wait profiling tool. This tool was based on a few special #define preprocessor commands along with __LINE__ and __FILE__ predefined macros. The macros wrapped pthread calls with a sequence of time measuring instructions and a call to a global recording object. As a result we got summarized time spend in each code line containing a pthread function. Calls to pthread_mutex_lock() were of the greatest interest to us.

Additionally, we measured the execution time of the program modifying the number of parallel threads. Serial execution provided us with timings of all "jobs" present in the program, which were to be distributed between threads in a parallel execution. The timings could tell us how much work could be dispatched to parallel threads and how much remained serialized.

3 The Case – Compression of Loaded Data

Infobright database uses a patented compression algorithm to obtain high data compression ratio, usually cited as 10:1, although some users claim to achieve ratio more than 30:1. All data is compressed. In result, several TB of source

data can be stored on a standard hard disk. Because the compression is a very important feature for Infobright and because Infobright targets databases (data warehouses) several TB big, Infobright data loader must deal with data compression very efficiently to provide high load speeds.

Infobright is a columnar database (see [12] for further references). The data loading process divides the source data into rows and each row into fields. Data items belonging to a particular column across the rows are assembled together to form a so called data pack. A data pack contains up to 64K values from a single column. To retrieve a single row from a table, constituent fields must be extracted from data packs from each column. E.g. row 100000th is assembled by taking 34464th item from the second data pack from each column. Of course, for the efficiency only columns used in a query are retrieved.

While loading, there is one data pack opened per each column. When a data pack gets full, it is compressed (each data pack is separately compressed) and stored on disk. A fresh data pack is created to accommodate subsequently loaded data items. For example, a source file containing 129000 rows, each row consisting of 3 fields will be loaded into (129000 / 65536 + 1) * 3 data packs.

The multithreaded version of the loader uses the main thread to read the source data, divide them into rows and columns, parse values and assemble them into data packs. For each data pack a thread is created and its task is to compress and save the data pack on disk. This solution worked quite well, however in some cases we observed that on a 4-way machine only 2-3 cores were efficiently used during the load process. We will come back to it later in the next section. For our experiments we decided to use a standard data set – the "part" table from the TPC-H test suite [13]. Because the load process performs uniformly as it progresses, for our convenience we took a relatively small sample of 3 200 000 rows.

3.1 Various Performance Related Areas

At first we confirmed, using OProfile-produced code execution profile (CPU_CLK_UNHALTED events), that thread creation and destruction, although performed fairly frequently (1 thread per 64K values), does not impose any significant overhead on Linux 2.6 running on an Intel x64 CPU – much less than 1 percent. Therefore we dropped an idea of introducing a thread pool.

The same execution profile showed that most of the computation time is spent in compression and initial parsing of the source data. It was no surprise as compression is known to be computationally intensive and the source files were in a text format (the most frequent case for Infobright users) requiring quite intensive processing. In Table 1we can see some details. The table shows for how many percent of the total CPU time a given function is responsible. The 3 first rows and the 5th row refer to compression related functions. Other rows refer to functions responsible for text parsing. We can calculate that the compression amounts to around 65% of the total computational effort (not to be confused with computation/elapsed time). Therefore one can expect that (by Amdahl's low, taking parsing as a sequential code) through delegating the

compression to parallel threads, it should be possible to obtain a 2.8 speedup for this case ("part" table). However, the practically observed speedup (elapsed time of serial vs multithreaded loading) for the test data reached 2.1 only. We needed to investigate why the loader could not reach the theoretical speedup limit. It was also important to learn how the speedup depends on the characteristics of load data sets and how the parallelization method can be improved.

The `vmstat` command has shown significant disk activity and CPU wait time during load. There was a possibility, that loading cannot proceed faster because of I/O bottleneck. Instead of careful analysis of vmstat output, we compared the load time when all the data had to be fetched from disk with the case when the whole source file had been preloaded into the Linux system disk cache. A command `echo 3 > /proc/sys/vm/drop_caches` has been used to clear the system disk cache. On the other hand, the source file has been small enough to fit completely in the disk cache after being read initially from the disk. Reading from disk turned out to be responsible for 24% slow down. At first we considered it as a very important factor. However, the measurements were taken on a usual PC with a single SAS drive 7200 rpm. In a server system this number should be smaller. On a dual Xeon server box equipped with an Adaptec RAID controller for the database storage and a Linux software RAID-0 for the source files, the slow down dropped to 4% only. Therefore disk access has been eliminated from the list of main factors limiting the speed of the load process. Moreover, the observable speedup in the case when source data had been preloaded was still around 2 only.

The loader uses a number of data shared between threads. It was crucial to learn if waiting for exclusive access to these shared data was causing significant delays. The output of our simple lock-wait profiling tool revealed that it was not the case. This finding, supported by application-level monitoring of the number of existing threads, has proved that the observed idling of some CPU cores was caused by an insufficient number of threads rather than thread waiting. The TPC-H "part" table has a few numeric columns and only 2 longer text (VARCHAR) ones. A compressing and saving thread is started for each assembled data pack for each column. By using our FET tool, we recorded the thread life times summarized separately for each column and we captured also the execution

Table 1. Counted CPU_CLK_UNHALTED events for sequential loading of "part" table

Samples	%	Symbol name
1384194	36.0123	IncWGraph::EncodeRec(unsigned
537732	13.9901	IncWGraph::Traverse(IncWGraph::Node*&,
469181	12.2066	FTree::GetEncodedValue(RCBString)
251769	6.5502	DataParserForText::GetRowSize(char*,
111539	2.9019	IncWGraph::Node::Duplicate(IncWGraph::Node*,
92712	2.4121	RSIndex_CMap::PutValue(RCBString&,
84231	2.1914	DataParserForText::ProcessEscChar(int)
78127	2.0326	DataParserForText::PrepareNulls()

times of compression procedures summarized for each column. We noticed, that the compression amounts for almost the whole thread lifetime. Text compression (columns 1 and 8 in the "part" table, see Table2) turned out to be orders of magnitude slower than numeric compression (other columns).

Comparing the compression time of a numeric column to the time spent on parsing and assembling data packs (FET and OProfile results), we concluded, that the time necessary to assemble a numeric data pack is not larger than the time necessary to compress and save it. Therefore, a single compressing thread is able to process data packs at the rate they are assembled. In other words, sequential parsing turned out to be a bottleneck in the case of numeric data, for which compression is quick. Here, two solutions could be proposed: a) optimization of the parsing procedure and b) reengineering of the loader logic, so that more work is shifted to parallel threads. We found both possibilities to be feasible; however their further discussion is out of the scope of this paper.

3.2 Cache Behavior

In out tests we used a machine with an Intel Core2 Quad (4 cores) 2.4GHz CPU equipped with 4MB of L2 cache. It should be possible to potentially achieve load speedup up to 4 on this processor and the maximal speedup 2.8 calculated for the considered case should be obtainable easily. As we stated above, only speedup around 2 was observed. To explain it, we studied the profiling results in more detail. The FET results for loading with one compression thread and with many parallel compression threads are presented in Table 2. The same procedure (compression) was taking up to 3 times more CPU time when many threads were working in parallel. This result seemed puzzling, because the compression procedure does not use any synchronization primitives, it does not access files and not all CPU cores were fully used. The computations were just going slower when many threads were active, even if the threads did not need to compete for CPU access.

The OProfile tool uses hardware performance counters available on modern CPUs to get a desired application performance profile. We decided make a num-

Table 2. CPU time used by column compression for single vs many parallel threads compressing simultaneously on Intel Core2 Quad

Column name	Time 1 thread	Time many threads
P_PARTKEY	0.11	0.16
P_NAME	16.92	26.92
P_MFGR	0.12	0.38
P_BRAND	0.12	0.17
P_TYPE	0.12	0.14
P_SIZE	0.12	0.19
P_CONTAINER	0.12	0.19
P_RETAILPRICE	0.10	0.20
P_COMMENT	5.50	9.12

Table 3. Counted L2 cache line misses (MEM_LOAD_RETIRED events)

Symbol name	#misses 1 thread	#misses many threads
IncWGraph::EncodeRec	5891	6822
IncWGraph::Traverse	2364	2698
DataParserForText::PrepareObjsSizes	527	574

ber of different profiles to get an answer why the compression performance was decreasing if more concurrent computational threads were active. The first guess was that cache L2 misses were much more frequent in multithreaded case, as the threads were competing for cache lines. The compression code caused most of the L2 misses and indeed, there was less L2 misses if only 1 compression thread was active, see Table 3. The shown numbers correspond to the number of cache misses which occurred during the load. However the difference between using one and many threads – around 17% - could not be entirely responsible for the observed computation slow down. Further profiling covered data cache L1 misses, here very small differences were discovered between single and multithreaded execution. Similarly DTLB (Data Translation Lookaside Buffer) misses presented only insignificant differences.

Table 4 top, reveals that multithreaded compression generated on average 1.66 times more memory transactions, than compression performed by a single thread. The large absolute number of memory accesses must have had a significant influence on the observed computational performance. It remained unclear why multithreaded compression needed to access the memory more often to compress the same amount of data. The answer came when we returned to L2 monitoring, but this time we observed how cache lines were removed from the cache. Lines are evicted to make room for currently needed data. If an evicted line had been modified, it must be written back to the main memory. On average 1.72 times more evictions happened for multithreaded compression, see Table 4, bottom. Apparently most of the evicted L2 lines were "dirty", triggering the

Table 4. Counted number of completed memory transactions (BUS_TRAN_MEM events, top) and number of L2 cache line evictions (L2_LINES_OUT events, bottom)

Symbol name	Samples 1 thread	Samples many threads
IncWGraph::EncodeRec	36854	58855
IncWGraph::Traverse	14069	23398
DataParserForText::PrepareNulls()	4876	7200
FTree::Add	2474	5697

Symbol name	Samples 1 thread	Samples many threads
IncWGraph::EncodeRec	30305	47519
IncWGraph::Traverse	11346	19238
DataParserForText::PrepareNulls()	4772	6015
FTree::Add	1896	4952

Table 5. Comparison of CPU time used by column compression using single vs using many parallel compressing threads on Intel 2x Xeon 3.2GHz with 2-way HyperThreading and on Core2 Quad

Column name	Time 1 thread Xeon	Time many threads Xeon	Time 1 thread Core2	Time many threads Core2
P_PARTKEY	0.22	0.72	0.11	0.16
P_NAME	32.40	37.90	16.92	26.92
P_MFGR	0.21	0.27	0.12	0.38
P_BRAND	0.23	0.32	0.12	0.17
P_TYPE	0.23	0.33	0.12	0.14
P_SIZE	0.23	0.26	0.12	0.19
P_CONTAINER	0.22	0.28	0.12	0.19
P_RETAILPRICE	0.28	0.33	0.10	0.20
P_COMMENT	12.43	16.29	5.50	9.12
SUM	**46.45**	**56.70**	**23.23**	**37.47**
Slowdown	**1.22**		**1.61**	

reported memory accesses. While proper cache usage proofs to be important for multithreaded computations, the software prefetching proposed in [5] cannot help in this case. The prefetching can work only for cases when one can predict what data will be needed in the near future, while compression accesses large memory structures randomly. Also, prefetching data needed by one thread could flush from the cache data used by another compressing thread.

All the results presented so far were taken on a Core2 Quad machine. For a comparison we employed a server equipped with 2 Xeon 3.2 GHz CPUs. The Xeons had HyperThreading extensions, making the whole the system to present itself as a 4-way machine, with 2MB of cache per CPU. Unfortunately, Intel Xeon/P4 performance counters are much different than in the case of Core2. Therefore we turned towards FET measurements. We noticed that in the case of Xeons multithreaded compression degraded the computational capacity of the CPU much less than in the case of Core2, see Table 5. On Xeons a single compression took on average 1.22 times more time if it was executed in parallel with other compressions, while on Core2 Quad this average slowdown reached 1.61. We think, that Xeons performance was more stable for two reasons: a) Xeons were slower in absolute terms (see the timings in Table 5, single-threaded compression took 46.45 seconds on the Xeon and only 23.23 seconds on Core2), so memory stalls were less expensive in terms of the amount of computations which could be done in a time of a stall, b) the differences in CPU architectures, especially cache, may play a role here.

4 Conclusions

Multithreaded programming is the clue to the power of modern multicore microprocessors. Unfortunately, the efficiency of a multithreaded program can be lost due to many factors. A parallel data loader performing data compression

is an example of an application potentially facing I/O problems in addition to performance problems common in multithreading, like serialization on mutexes. We presented a rather complete list of issues influencing the loader performance, along with a set of tools necessary to deal with those issues. We found that proper performance debugging requires a fair number of experiments and compels a programmer to employ various methods and tools in an orchestrated way. We managed to identify two main factors limiting the performance of the multithreaded data compression: a) serial parsing too slow in comparison to the compression of numerical data and b) a degradation of the CPU computational performance due to much increased memory access rate caused by the competition for the cache. We noticed that the degradation is much more visible on Core2 Quad than on a Xeon system. We plan to investigate more the architectural differences leading to the observed performance variations, including also AMD processors. We will also verify the loader performance for other data sets, especially for data with more text columns. This way we will limit the influence of the serial parsing and allow for more speedup, possible exposing more interesting differences between various CPUs.

References

1. AMD, http://multicore.amd.com/us-en/AMD-multi-core/multi-core-advantage.aspx
2. Intel, http://www.intel.com/technology/architecture/downloads/quad-core-06.pdf
3. Gepner, P., Fraser, D.L., Kowalik, M.F.: Performance evolution and power benefits of cluster system utilizing quad-core and dual-core intel xeon processors. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 20–28. Springer, Heidelberg (2008)
4. Tao, J., Kunze, M., Karl, W.: Evaluating the cache architecture of multicore processors. In: Proc. of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing, PDP 2008, pp. 12–19. IEEE, Los Alamitos (2008)
5. Papadopoulos, K., Stavrou, K., Trancoso, P.: HelperCoreDB: Exploiting multicore technology for databases. In: 16th International Conference on Parallel Architecture and Compilation Techniques PaCT 2007 (2007)
6. Hardavellas, N., Pandis, I., Johnson, R., Mancheril, N.G., Ailamaki, A., Falsafi, B.: Database servers on chip multiprocessors: Limitations and opportunities. In: Proceedings of the Biennial Conference on Innovative Data Systems Research (2007), http://www.cidrdb.org/
7. Poess, M., Potapov, D.: Compression in oracle. In: VLDB 2003, pp. 937–947 (2003)
8. Holloway, L., Raman, V., Swart, G., DeWitt, D.J.: How to barter bits for chronons: Compression and bandwidth trade offs for database scans. In: SIGMOD Conference 2007, pp. 937–947 (2007)
9. Infobright: http://www.infobright.org, www.infobright.com
10. OProfile - a system profiler for linux, http://oprofile.sourceforge.net
11. Valgrind, http://valgrind.org/
12. Slezak, D., Wroblewski, J., Eastwood, V., Synak, P.: Brighthouse: An analytic data warehouse for ad-hoc queries. In: Proceedings of the VLDB Endowment, vol. 1(2), pp. 1337–1345 (2008)
13. Transaction Processing Performance Council: http://www.tpc.org/tpch/

Energy Considerations for Divisible Load Processing

Maciej Drozdowski*

Institute of Computing Science, Poznań University of Technology,
Piotrowo 2, 60-965 Poznań, Poland
Maciej.Drozdowski@cs.put.poznan.pl

Abstract. In this paper we analyze energy usage in divisible load processing. Divisible load theory (DLT) applies to computations which can be divided into parts of arbitrary sizes, and the parts can be independently processed in parallel. The shortest schedule for divisible load processing is determined by the speed of computation and communication. Energy usage for such a time-optimum schedule is analyzed in this paper. We propose a simple model of energy consumption. Two states of the computing system are taken into account: an active state and an idle state with reduced energy consumption. Energy consumption is examined as a function of system parameters. We point out possible ways of energy conservation. It is demonstrated that energy can be saved by use of parallel processing.

Keywords: Energy-efficient computing, performance evaluation, divisible loads.

1 Introduction

Divisible load theory (DLT) is a new parallel processing paradigm applicable in computations which can be divided into parts of arbitrary sizes and processed independently on remote computers. In other words, the DLT assumptions are relevant to computations with fine granularity and negligible data dependencies. Processing big volumes of data is an example of divisible computation. Consider searching for patterns in medical screening photographs. The set of photographs can be partitioned with granularity of one picture. If the number of pictures is big, then the resolution of partitioning the whole dataset is fine. The photographs can be analyzed independently of each other. Other examples of divisible computations include processing measurement data (e.g. SETI@home), image and video processing, linear algebra, search for combinatorial objects (e.g. distributed.net). Divisible load theory originated in the late 1980s [1,3] as a way to strike a compromise between the communication delays and the gains from faster parallel processing. Surveys of DLT, and its practical applicability can be found in [2,4,7].

* Partially supported by grant N N519 1889 33 of Polish Ministry of Science and Higher Education.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 92–101, 2010.

Energy consumption of contemporary data centers, and supercomputing facilities is becoming a limiting factor to their growth [6]. Buying power is becoming more and more expensive. For example, the average power usage of the first 5 supercomputers from the current (June 2009) top500 list [9] is over 3.2MW. At the current prices in Poland (≈0.3PLN/kWh) this would cost ≈23.3k PLN daily, and ≈8.5M PLN annually (resp. ≈5.5k, ≈2M EUR). Thus, careful and economic use of energy is an indispensable element of the future high performance computing.

The cost of divisible load processing in general has been analyzed in [5,8]. The cost considered in [5,8] could be monetary as well as energy. In both publications two problems were analyzed: to find the minimum cost schedule for a given schedule length, or to find minimum schedule length for a given cost limit. Both papers took into account the cost of computation only. In [8] the sequence of activating heterogeneous computers minimizing the cost was proposed, but computation startup costs were ignored. In [5] it was shown that the above problems are computationally hard (strictly **NP**-hard) when the startup time is non-negligible. Yet, for a given sequence of communications between the processors and the load distributor the problem can be solved by reduction to linear programming. In this paper we assume that performance is the primary criterion. Therefore, the shortest schedules are used. Energy consumption is examined for such time-optimum schedules. Moreover, we concentrate on a more energy-specific representation of the costs of load processing. It is assumed that both the communication and the computations use energy. The costs of computation initiation (startup costs) are taken into account. Similarly to [11] we assume that computing system can be in two states: idle and active. In the idle state power usage is reduced.

The rest of this paper is organized as follows. In the next section we formulate the mathematic model of divisible load processing both with respect to the timing and to the energy cost. In section 3 results of performance evaluation are presented. In section 4 we provide conclusions and discuss lessons learned.

2 Problem Formulation

In this section we outline construction of the optimum length schedule, as well as its energy consumption. Words computer, processor will be used interchangeably. A processor consists of a CPU, memory, and network interface. The CPU and the network hardware can work in parallel such that simultaneous communication and computation is possible. The topology of the processor interconnection is a star (a.k.a. single level tree). In the center of the star resides a processor P_0 called originator (also called master, server) which distributes the load to the remaining processors $P_1, \ldots, P_m$ (called slaves, workers). The star topology may represent a computer cluster in a local area network or a set of computers interconnected in the grid infrastructure. The computing environment is homogeneous.

Timing Model. It is assumed that initially volume V of load resides on the originator. The load is sent from the originator to processors $P_1, \ldots, P_m$ in pieces

of sizes $\alpha_1, \ldots, \alpha_m$. To receive a piece of load a processor has to be *activated* first. The activation process may include transition of the computer hardware and software from the idle to the running state, loading the divisible application runtime environment (such as virtual machines and libraries), starting the application, allocating memory and bringing the application to the state of active waiting for the message with the piece of load. The time of activation is denoted S, and referred to as computation startup time. Sending α_i units of load to processor P_i takes time $\alpha_i C$. Computations for this amount of load take time $\alpha_i A$. It is assumed that the time of returning the results to the originator is very short and can be neglected. It is a common assumption in DLT made for the sake of simplicity in mathematical modeling [2,4,7]. We distinguish two cases depending on the participation in the computations of the originator. If the originator is dedicated solely to distributing work then it receives no load (Fig.1a). Otherwise, originator takes part in the computation and processes load $\alpha_0 > 0$ (Fig.2a).

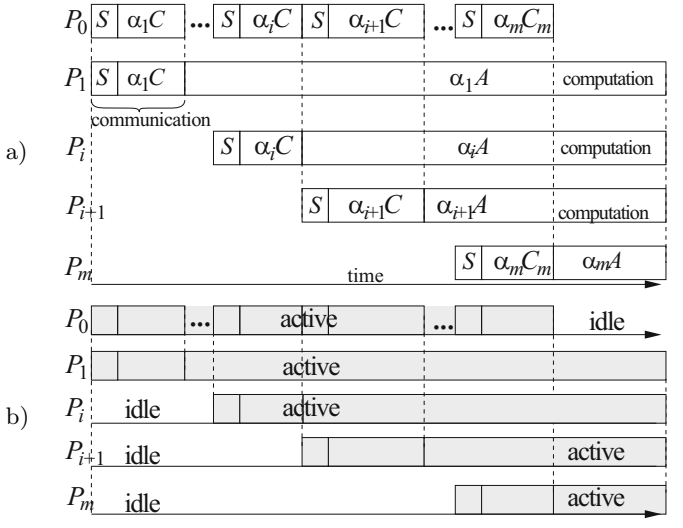


Fig. 1. Initiator *is not* computing. a) Communication and computation schedule. b) Power usage schedule.

The optimum schedule length is obtained by selecting sizes of load chunks $\alpha_1, \ldots, \alpha_m$, and α_0 if applicable. It has been shown in [2] that if the result returning time can be neglected, then for the minimum schedule length all processors must stop computations simultaneously.

Assume that the originator is not computing. The above observation leads to the system of linear equations, from which chunk sizes are derived (cf. Fig.1a):

$$A\alpha_i = S + (C + A)\alpha_{i+1} \quad \text{for} \quad i = 1, \ldots, m-1 \tag{1}$$

$$\sum_{i=1}^{m} \alpha_i = V \tag{2}$$

Let us denote as $\sigma = S/A$, and $\rho = 1 + C/A$. Then,

$$\begin{aligned}\alpha_i &= \sigma + \rho\alpha_{i+1} = \\ &= \sigma + \rho(\sigma + \rho\alpha_{i+2}) = \sigma + \rho\sigma + \rho^2\alpha_{i+2} = \ldots \\ &= \sigma + \rho\sigma + \ldots + \sigma\rho^{m-i-1} + \rho^{m-i}\alpha_m = \\ &= \frac{\sigma(1-\rho^{m-i})}{1-\rho} + \rho^{m-i}\alpha_m. \end{aligned} \tag{3}$$

for $i = 1, \ldots m$. From (2) and (3) we obtain

$$\begin{aligned} V = \sum_{i=1}^{m} \alpha_i &= \sum_{i=1}^{m} \left(\frac{\sigma(1-\rho^{m-i})}{1-\rho} + \rho^{m-i}\alpha_m \right) = \\ &= \frac{\sigma}{1-\rho}\left(m - \frac{1-\rho^m}{1-\rho} \right) + \frac{\alpha_m(1-\rho^m)}{1-\rho}. \end{aligned} \tag{4}$$

Consequently,

$$\alpha_m = \frac{V(1-\rho)}{1-\rho^m} - \frac{\sigma(m(1-\rho) - 1 + \rho^m)}{(1-\rho^m)(1-\rho)} \tag{5}$$

Note that in the above equation we have subtraction, and α_m may become negative if σ and m are sufficiently big. All such combinations of V, m, C, S, A that $\alpha_m < 0$ are rejected as infeasible. In practice $\alpha_m < 0$ means that the load V is too small to employ all m processors for the current communication parameters C, S and processing rate A. If the programmer still decided to use this number or more processors, then schedule length would grow instead of decreasing. Consequently, efficiency of the schedule would also unnecessarily decrease.

When the originator is not computing schedule length is

$$T(m) = S + (C + A)\alpha_1, \tag{6}$$

where α_1 is calculated from (3) and (5).

If the originator is computing (Fig.2a) then partitioning of the load can be derived in the same way as in the previous case, however, equations (1) and (2) start from $i = 0$. Equation (3) is valid for $i = 0, \ldots, m$. Analogously to (4), (5) we obtain

$$\alpha_m = \frac{V(1-\rho)}{1-\rho^{m+1}} - \frac{\sigma(m(1-\rho) - \rho + \rho^{m+1})}{(1-\rho^{m+1})(1-\rho)}. \tag{7}$$

Schedule length is

$$T(m) = S + A\alpha_0, \tag{8}$$

where α_0 is calculated from (7) and (3).

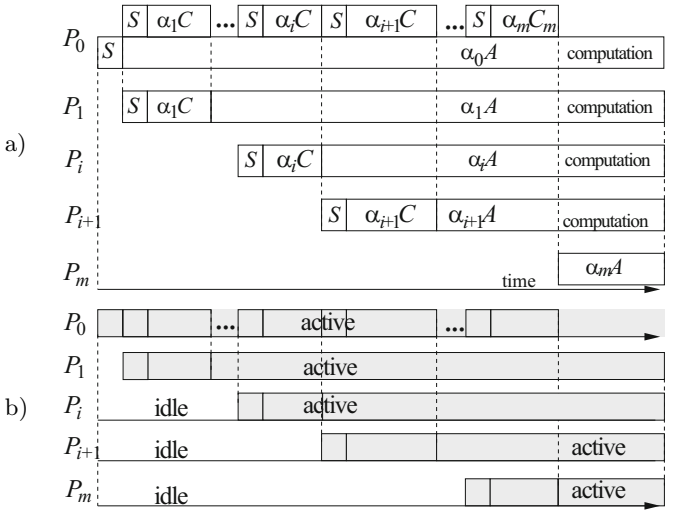


Fig. 2. Initiator *is* computing. a) Communication and computation. b) Power usage.

Energy Use Model. Now let us analyze the energy usage. Both the network and the processors may be either active or idle. It is assumed that active processors consume power P_C, and the active network equipment consumes power P_N. In the idle state power consumption is k times smaller, i.e. P_C/k, and P_N/k, for processors and for the network, respectively. To simplify mathematical formulae we divide the energy usage into two parts: the idle state energy, and the energy beyond the idle state consumed when processors and the network are running. During the whole schedule of length $T(m)$, the originator, m idle processors and the idle network consume energy

$$E_I = T(m)((m+1)P_C + P_N)/k. \tag{9}$$

The network is in the running state at the beginning of the schedule while distributing the load (cf. Fig.1b, and Fig.2b). Suppose the originator is not computing. Processor activation and the load distribution time is $\sum_{i=1}^{m}(S + C\alpha_i) = mS + CV$. The energy consumed above the network idle state is

$$E_{RN} = P_N \frac{k-1}{k}(mS + CV). \tag{10}$$

The originator is active during the whole load distribution time $mS + CV$, which results in energy consumption $P_C \frac{k-1}{k}(mS + CV)$. The remaining processors switch from the idle state to the running state when they are activated. Thus, processor P_i is active for $S + \alpha_i(C + A)$ units of time, consuming energy $P_C \frac{k-1}{k}(S + \alpha_i(C + A))$ above the idle state. The total computation energy consumption beyond the idle state is

$$E_{RC} = P_C \frac{k-1}{k} \left(mS + CV + \sum_{i=1}^{m} (S + (C+A)\alpha_i) \right)$$
$$= P_C \frac{k-1}{k} (2mS + (2C+A)V). \quad (11)$$

Suppose the originator is computing. The time of communications is $\sum_{i=1}^{m}(S + C\alpha_i) = mS + C(V - \alpha_0)$. The energy consumed by the network beyond the idle state is

$$E_{RN} = P_N \frac{k-1}{k}(mS + C(V - \alpha_0)). \quad (12)$$

The processors together consume beyond the idle state

$$E_{RC} = P_C \frac{k-1}{k} \left(S + A\alpha_0 + \sum_{i=1}^{m} (S + (C+A)\alpha_i) \right) =$$
$$= P_C \frac{k-1}{k} \left((m+1)S + VA + (V - \alpha_0)C \right). \quad (13)$$

The total energy consumed in the computation is

$$E = E_I + E_{RN} + E_{RC}. \quad (14)$$

3 Performance Evaluation

In this section we analyze the amount of energy E necessary to achieve certain schedule length $T(m)$.

Before presenting the details of the simulations let us examine a general relationship between the system and application parameters A, C, S, P_N, P_C, V, the number of used processors m, processing time $T(m)$, and energy E. As mentioned in the previous section the number of processors m that can be exploited depends on A, C, S, V. A general tendency in divisible load processing is that with growing C, S the number of usable processors decreases because communication delays increase and preclude effective use of many processors. On the other hand, with growing A, V the number of usable processors increases because relative contribution of communication delays to the schedule length decreases [2,4,7]. Since the reduction in processing time $T(m)$ comes from applying more processors, and the number of usable processors is limited, also the reductions in $T(m)$ are limited. Increasing C, S results in narrower range of processor numbers m where $T(m)$ is reduced. Conversely, increasing A, V widens the range of $T(m)$ reductions. Note that in the following charts $T(m)$ will be shown on the horizontal axis. Now let us examine energy as determined by equations (9) - (13). Intuitively, it can be expected that shorter schedules engage more processors, and hence, should be more costly in energy. Indeed, in all the above equations energy consumption grows with the processor number m. Beyond m, energy consumption depends on constants V, A, C, S, P_C, P_N, k. Optimizing them for minimum power usage is beyond the scope of this paper. Let us now proceed to the results of the simulations. In the

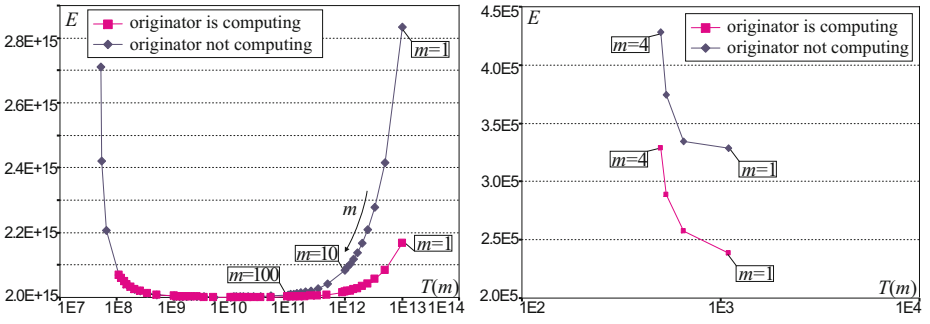
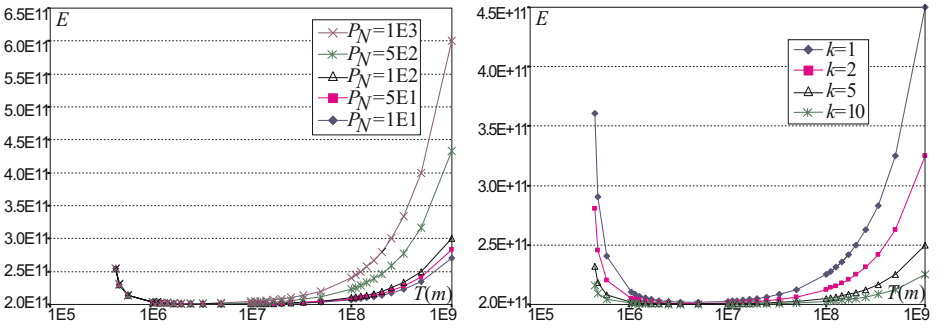


Fig. 3. Energy E vs. schedule length $T(m)$ for $A = 1, C =$ 1E-6, $S =$ 1E2, $P_N = 50, P_C = 200, k = 3$. a) $V =$ 1E13, b) $V =$ 1E3.

following figures we present dependence of the total energy used as defined in (14) versus schedule length defined in equations (6), or (8).

In Fig.3 energy consumption vs. processing time is shown. Values of the parameters used in Fig.3 can be interpreted as follows. Processing one unit of load takes 1s ($A = 1$), transferring it from the originator to the remote processor takes 1μs ($C =$ 1E-6), computation startup time is 100s, the network equipment uses only 50W of power in the active state ($P_N = 50$), a computing processor uses 200W of power ($P_C = 200$), in the idle state power usage is three times smaller ($k = 3$). Let us remind that $T(m)$ is not a real independent variable, because both $T(m)$ and E change as a result of using more processors m. Surprisingly, E as a function of $T(m)$ has a minimum. With increasing processor number execution time is decreasing, as could be expected, but initially also the energy used is decreasing. This behavior of E dependence on $T(m)$ can be explained by several phenomena. Let us assume that the originator is not computing. Note that in (9) the idle state energy depends on $T(m)$. With growing processor number m, execution time $T(m)$ decreases. Therefore, E initially decreases with decreasing $T(m)$. Most of this reduction can be attributed to shorter network and initiator idle state. The relative difference between the highest and the lowest energy consumption in the above experiments ranged from 30% to 40% (originator is not computing). The extent of energy savings may be surprising, considering their source. However, it is a result of long computation time when the communication system remains idle. On the other end of the diagram E is not growing to the infinity because increasing m leads to $\alpha_m < 0$ in equation (5) which means that it is impossible to activate all the processors with the given V. As noted in the previous section we reject such cases as infeasible. A wide plateau of energy usage in Fig.3a results from approximately equal effect of decreasing $T(m)$ in (9) and increasing component mS in (10), (11). With decreasing problem size V the interval of $T(m)$ with nearly flat energy usage narrows until disappearing completely for $V =$ 1E3, as shown in Fig.3b. The above observations apply also if the originator is computing, though this case is more energy efficient.



a) b)

Fig. 4. Energy E vs. $T(m)$ a) for changing P_N at $k = 3$ b) for changing k at $P_N = 50$, and $A = 1, C =$ 1E-6, $S =$ 1E2,$P_C = 200, k = 3$. Originator is not computing.

Consequently, energy savings are smaller. We further discuss the difference between the situation when the originator is computing, or not computing, at the end of this section.

In Fig.4a energy consumption vs. processing time is shown for various values of the network power usage P_N when originator is not computing. The bigger P_N is, the bigger the initial decrease of E with decreasing $T(m)$. On the other hand, when m is big, and $T(m)$ is near its minimum, E_{RC} is dominating in E, and all the functions end overlapping.

In Fig.4b energy consumption vs. processing time is shown for various values of the active to idle power usage ratio k. For instance, $k = 1$ represents the situation when power usage in idle state is no different than in the running state. This means that the whole energy consumption is described in equation (9). As it can be seen the biggest reduction in energy consumption takes place just for $k = 1$ which confirms that the energy savings result from shortening of the idle state. On the other hand, for $k > 1$ the energy savings are shallower in Fig.4b, but the total energy consumption is smaller than for $k = 1$. Let us note that one should not be confused that $k = 1$ is better than $k > 1$ because deeper energy reductions are not the same as smaller overall energy use.

When the originator is computing the dependencies of E on $T(m)$ for changing P_N, k are very similar. Therefore we do not present them here.

Let us now return to the difference between the cases when originator is and is not computing. By subtracting (12) from (10) we obtain the difference in the energy used by the network: $C\alpha_0 P_N(k-1)/k$. Analogously, from (11) and (13) the difference in the energy used by the running processors is $((m-1)S+C(V+\alpha_0))P_C(k-1)/k$. The total difference in energy use is

$$\Delta E = P_N \frac{k-1}{k} C\alpha_0 + P_C \frac{k-1}{k}((m-1)S + C(V+\alpha_0)). \tag{15}$$

Startup times mS cannot dominate in the processing time because otherwise distributed processing would be counterproductive. Hence, $(m-1)SP_C(k-1)/k$

does not constitute a big difference. The remaining components are related to CV and $C\alpha_0$. These gains are especially noticeable if C is big, e.g. $C \approx A$. Moreover, if the originator is computing it is possible to save energy by not sending load α_0 for remote processing. In this case we have an additional computer which nearly immediately starts processing the load. The two cases are juxtaposed in Fig.5. It confirms that the difference between the two cases is big when $C \approx A$. For example, for $C = 0.5$ the difference in energy used is in the range of 130%, while for $C =$ 1E-3 it is not more than 30%.

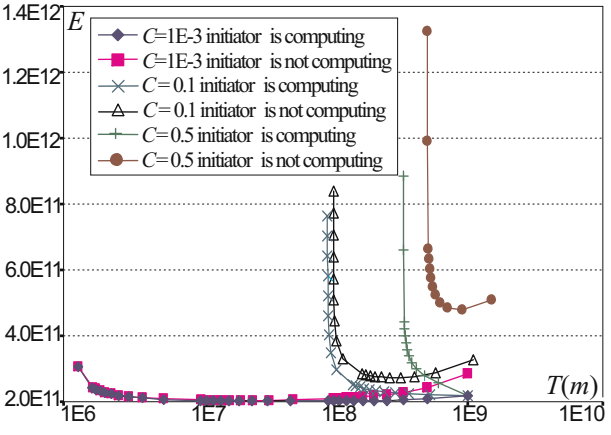


Fig. 5. Energy E vs. schedule length $T(m)$ for $A = 1, S =$ 1E2, $V =$ 1E9, $P_N = 50, P_C = 200, k = 3$ and changing C

Let us observe that Fig.5 also demonstrates influence of communication rate C on potential energy savings. As it can be seen, for small C energy consumption initially decreases with increasing m, and hence decreasing processing time $T(m)$. The plateau of nearly flat energy consumption spans three orders of magnitude in $T(m)$. On the other hand, for big C energy consumption decreases only marginally, and then quickly grows with m (while $T(m)$ is nearly constant). It can be concluded that small C is essential for allowing reduction in energy consumption. It means that bandwidth must be high. This condition coincides with the requirements for effective communication in parallel applications.

4 Conclusions

In this paper we analyzed energy use in distributed processing of divisible loads. The energy consumed has been presented as a function of the execution time. Surprisingly, it appeared that this function has a minimum, and with decreasing processing time energy used is also decreasing. Hence, we have demonstrated that it is possible to save energy by parallel processing. We compared two ways of processing divisible loads: with and without computations on the load originator.

It turns out that using the originator is more energy-efficient. Yet, the differences are apparent only if communication medium is slow.

Our analysis reveals that the savings come from shorter idle state of the communication subsystem. The network idle time is specific to divisible load processing. Similar idle intervals exist in other parallel processing models, e.g., bulk-synchronous processing [10]. Hence, also in other types of parallel applications reduction in network energy consumption should be possible. On the other hand, parallel applications which are communication intensive would have no such network idle time. Consequently, this kind of energy saving would not materialize. The analysis conducted in this paper points to a new way of economizing on energy which is often overlooked. Namely, communication network consumes energy, and also here considerable resources can be saved. Possibly, further savings may be achieved by grouping communications, and switching off the network when it is idle.

References

1. Agrawal, R., Jagadish, H.V.: Partitioning Techniques for Large-Grained Parallelism. IEEE Transactions on Computers 37, 1627–1634 (1988)
2. Bharadwaj, V., Ghose, D., Mani, V., Robertazzi, T.: Scheduling divisible loads in parallel and distributed systems. IEEE Computer Society Press, Los Alamitos (1996)
3. Cheng, Y.-C., Robertazzi, T.G.: Distributed computation with communication delay. Transactions on Aerospace and Electronic Systems 24, 700–712 (1988)
4. Drozdowski, M.: Scheduling for Parallel Processing. Springer, London (2009)
5. Drozdowski, M., Lawenda, M.: The combinatorics of divisible load scheduling. Foundations of Computing and Decision Sciences 30, 297–308 (2005)
6. Katz, R.H.: Tech titans building boom. IEEE Spectrum 46(INT), 36–49 (2009), http://www.spectrum.ieee.org/feb09/7327
7. Robertazzi, T.: Ten reasons to use divisible load theory. IEEE Computer 36, 63–68 (2003)
8. Sohn, J., Robertazzi, T.G., Luryi, S.: Optimizing computing costs using divisible load analysis. IEEE Transactions on Parallel and Distributed Systems 9, 225–234 (1998)
9. TOP500 List (June 2009), http://top500.org/list/2009/06/100
10. Valiant, L.: A bridging model for parallel computation. Communications of the ACM 33, 103–111 (1990)
11. Woo, D.H., Lee, H.-H.S.: Extending Amdahl's law for energy-efficient computing in the many-core era. IEEE Computer 41, 24–31 (2008)

Deskilling HPL

Using an Evolutionary Algorithm to Automate Cluster Benchmarking

Dominic Dunlop, Sébastien Varrette, and Pascal Bouvry

CSC research unit, University of Luxembourg, Luxembourg
Firstname.Lastname@uni.lu

Abstract. The High-Performance Linpack (HPL) benchmark is the accepted standard for measuring the capacity of the world's most powerful computers, which are ranked twice yearly in the Top 500 List. Since just a small deficit in performance can cost a computer several places, it is important to tune the benchmark to obtain the best possible result. However, the adjustment of HPL's seventeen configuration parameters to obtain maximum performance is a time-consuming task that must be performed by hand. In a previous paper, we provided a preliminary study that proposed the tuning of HPL parameters by means of an Evolutionary Algorithm. The approach was validated on a small cluster. In this article, we extend this initial work by describing ACBEA, a fully-automatic benchmark tuning tool that performs both the configuration and installation of HPL followed by an automatic search for optimized parameters that will lead to the best benchmark results. Experiments have been conducted to validate this tool on several clusters, exploiting in particular the Grid'5000 infrastructure.

1 Introduction

Statistics concerning high-performance computers are of major interest to manufacturers, users, and potential users. The Top500 project [2] operates at a worldwide level as a reference contest to evaluate the 500 most powerful computer systems. The list is updated twice a year and the computers are ranked by their performance on the long-established High-Performance LINPACK (HPL) [17] benchmark, despite the existence of newer alternative benchmarks [7]. HPL is a software package that solves a (random) dense linear system using double-precision (64 bit) floating-point arithmetic on distributed-memory computers. Seventeen configuration parameters should be tuned and adapted to the computing platform to obtain maximum performance. Even though some guidelines exist to guide the search of the parameter space (firstly from the authors of HPL themselves, and secondly in articles that discuss HPL tuning such as [5,19]), this is generally a tedious task that is performed by hand. In a previous paper [6], we showed how an evolutionary algorithm (EA) may be exploited to determine the

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 102–114, 2010.

best possible parameters in a nearly automatic way, in order to maximize the results of the benchmark. The approach was validated on a small cluster hosted at the University of Luxembourg.

In this article, we describe the extension of this approach into a framework called ACBEA (*Automatic Cluster Benchmark with Evolutionary Algorithm*), which provides a fully-automatic benchmark tuning tool based on an EA that explores the parameter space with many small benchmark runs, delivering parameter combinations that are likely to produce outstanding results in larger runs. The approach may be used iteratively if necessary, progressively reducing the proportion of the parameter space explored.

2 Context and Problem Statement

HPL [17] solves a dense N by N system of linear equations $A \times x = b$ (divided into blocks of size $P \times Q$) by Gaussian elimination with partial pivoting. As well as N, P and Q, fourteen further parameters control HPL's execution, and any system administrator who has tried to evaluate the computing power of a cluster with HPL can testify to the difficulty of manually tuning these parameters to maximize the benchmark result. The problem is due to the size of the search space and the fact that a single run can take more than half a day.

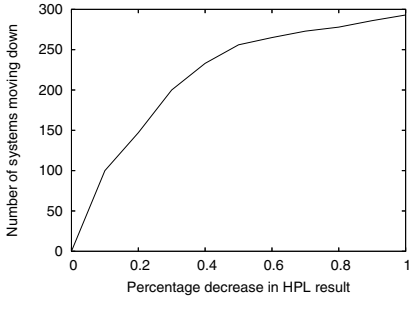


Fig. 1. Impact of HPL result reduction on the Top500 rank

That this tuning is of crucial importance is illustrated by figure 1, which shows how many systems in the November 2009 Top 500 list [2] would lose one place or more if their HPL result were slightly lower.

Our previous work [6] promoted the idea that HPL can be seen as an *objective function* for an Evolutionary Algorithm (EA) in such a way that it faciliates and automates the tuning process. EA refers to a class of problem-solving techniques based on the Darwinian theory of evolution. A possible and acceptable solution *i.e.* a member of the *population* is called an *individual*. Each iteration (or *generation*) of an EA involves a set of genetic operators randomly applied to the individuals together with a competitive selection that weeds out poor individuals through the evaluation of a *fitness value* that indicates their quality as a solution to the problem. More details on EAs may be found in [10]. The EA in [6] is configured as follows. An individual corresponds to a set of eligible parameters for HPL. Its fitness value is the benchmark result when running HPL on the cluster with those parameters. Our initial study delegated the details of the evolutionary computation to ACOVEA, a framework initially designed to investigate the optimum combination of command-line flags for a compiler (ACOVEA stands for *Analysis*

of Compiler Options via Evolutionary Algorithm). Using an adapter to match HPL to the ACOVEA interface when benchmarking a small cluster, spectacular results were obtained with little effort compared to classical hand-tuning.

This paper extends our initial proposal in two directions. Firstly, the details of the evaluation process to find the most suitable library are set out (see §3.1). Secondly, we describe an all-in-one framework called ACBEA (see §3.2) for benchmarking the computing power of a cluster through HPL. This tool is designed to download, build and launch HPL in such a way that the process of parameter tuning is handled internally and sequentially, starting from a small problem size and moving to the largest possible. Hopefully, this last configuration will produce the best benchmark result for the computing platform. ACBEA makes use of an EA to automate nearly all tedious processes such that the interaction of the user is limited to an initial setup, some manual tuning for the last step of the evaluation and finally the collection of the ultimate result (see §3.3). As before, our approach is based on the assumption that individuals that produce good results in small, short benchmarks are likely to produce good results in larger, longer tests. This hypothesis follows from practical observations and is discussed in §4.

3 Acbea Software Components

The software harness used in [6] was assembled quickly using scripting tools. As such, it was difficult to run and to maintain, and suffered from a number of inefficiencies. For example, the evaluation of each set of HPL parameters required a batch job to be submitted to start a new instance of HPL on the cluster's compute nodes. Thus each evaluation incurred both batch submission and HPL start-up overhead. For the follow-up work documented here, a more flexible, efficient and maintainable package was developed from the ground up. The package was designed with several constraints in mind. First of all, to have a *maximal portability*: the software package should build and run in as many UNIX-like environments as possible, and be able to utilize a choice of components for the following elements:

- C and C++ compilation systems. GCC and HP's aC++ were used in development, but other products such as Intel's icc can also be used.
- Batch job submission system. Development has been carried out exclusively with OAR [4], but hooks are provided to allow alternative schedulers.
- BLAS (Basic Linear Algebra Subroutines) library implementation. We use ATLAS [21] as a default, but alternative implementations can be used.
- Message Passing Interface. OpenMPI [8] was used in most tests, but alternatives are supported.

We also want to ensure *minimal prerequisites*, *liberal licence terms*, and finally *no modification to HPL source code.*

3.1 Choice of Evolutionary Algorithm Library

Thirteen evolutionary algorithm library packages, all written either in C or C++, were evaluated against five criteria:

1. *Portability.* The packages were built in four environments: FreeBSD, HP/UX, Linux and Mac OS X. Packages that built and passed their own test suites in all environments were given a higher score. Points were deducted if the build process was difficult and/or required additional packages.
2. *MPI support.* Two points were given to packages that included support for MPI.
3. *Currency.* Packages having a recently-released revision were marked higher than those that had not been updated for some time. The thinking behind this was that a recent revision suggested the existence of an active development community that would be able to provide support if necessary.
4. *Maturity.* The initial release date and the revision history of each package were examined to judge its maturity. Packages that had been available for several years and which had been regularly updated were marked higher than new packages, or old packages that had seen few revisions.
5. *Size.* Small packages were marked higher than large. It should be noted that the large packages support a wide variety of heuristic optimization methods. However, as it was not the aim of the research described here to test alternative methods, this was not considered an advantage.

Table 1. EA library evaluation

Package	***Ver.***	***Date***	***Good builds***	***Bundle size***	Portability	MPI support	Currency	Maturity	Size	***Total***
Evocosm [13]	3.3.1	2008	5	532kB	4	0	5	5	4	**18**
GAlib [20]	2.4.7	2007	5	368kB	4	0	4	4	4	**16**
Open BEAGLE [9]	3.0.3	2007	5	4.8MB	4	0	4	5	3	**16**
PGAPack [15]	1.1	2008	5	548kB	4	2	3	3	4	**16**
EO [11]	1.0.1	2008	4	972kB	3	0	5	3	4	**15**
GAtoolbox [18]	*n.a.*	2007	4	40kB	3	0	4	2	5	**14**
ParadisEO [3]	1.1	2008	4	20.5MB	3	2	5	3	0	**13**

The result of the evaluation for the seven highest-scoring packages is shown in table 1. The Evocosm [13] package scored highest, and so was chosen as a basis for Acbea. Evocosm implements a classical evolutionary algorithm as described in [10]: individual experiments are described by a genome made up of genes representing parameter values for the experiment; genomes that produce good experimental results are more likely to be used in creating the genomes used in the next generation than those that produce poor results. Each individual in the next generation is created by choosing two parents, and selecting each gene in the new individual from one of the parents at random[1]. Individual genes may

[1] This differs from the classical concept of *crossover* in that no attempt is made to preserve groups of genes that are adjacent to one another.

also mutate to a value that differs from either of the parent genes. Optionally, an elitist strategy may be used to preserve the best individuals. Additionally, Evocosm implements an island model *i.e.* it maintains several populations that exchange some individuals periodically. Note that this library also underlies the Acovea [12] framework used in our earlier work.

3.2 Acbea

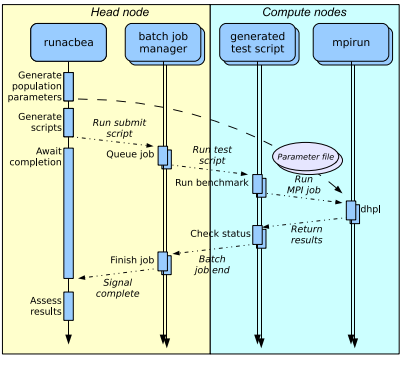


Fig. 2. ACBEA: population evaluation

ACBEA consists of a suite of programs that work together to automate the benchmark process. The most important of these is `runacbea`, which runs on the head node of a cluster and submits batch jobs for the cluster's compute nodes. The jobs are typically handled by a batch job manager.

`Runacbea`'s XML-format configuration file describes HPL's parameters and their allowable values. It also contains information about the batch job manager and the implementation of MPI that is to be used.

The program's operation for a single population of benchmark evaluations is shown in figure 2. The sequence of operations is repeated for each population in a generation, and the overall sequence is repeated until a specified number of generations has been run. If sufficient compute nodes are available, the task of fitness assessment for each population may be shared among several parallel jobs, so speeding evaluation. Each batch of benchmarks is run using MPI to launch multiple copies of `dhpl`, a customized variant of HPL' s `xhpl` benchmark program. While `xhpl` uses a short configuration file to describe a series of related tests, `dhpl` uses a file of arbitrary length to define the series of unrelated tests that represents all or part of a population. Conforming to the constraints presented in §3, the HPL problem solution code is unchanged. The result output format has been changed as little as possible. On terminating, `runacbea` summarizes its findings and produces a number of output files. The first contains a configuration for a subsequent run with a problem double the size on four times the number of cores. As four times the compute power is being applied to a problem having eight times the complexity, each benchmark will take almost twice as long as those defined by the original configuration file. In order that the subsequent run may explore only the more profitable parts of HPL's parameter landscape, the parameter values allowed by the new configuration file exclude those which appear only in most poorly-performing 33% of individuals in the run. (This cut-off level may be changed.) The remaining outputs are configuration files for `xhpl`, representing the parameters that produced the best-performing individual in the each population of the final generation. These files may be used to run `xhpl` benchmarks directly. The decision to host `runacbea` on the head node of a cluster may be questioned, as the intention is to

benchmark the compute nodes, while the main task of the head node should be to run administrative housekeeping functions for the cluster. In fact, `runacbea` may itself be viewed a housekeeping program: tests show that it and its child processes consume perhaps five seconds of processor time over an entire run, during which the compute nodes may clock up hundreds of hours.

3.3 The Benchmarking Process

The benchmarking process with ACBEA involves the following steps:

1. Gather information about the target cluster: nodes, cores and memory per node, MPI implementation, batch job manager ...
2. Use the provided `ten-sec-n` utility to obtain a value of N that makes HPL run for ten seconds on a single core . Let N_{ten_sec} be this value.
3. Edit the `runacbea` configuration file to create one suitable for testing all the cores in a small group of nodes n — four has been found to be a reasonable choice for n. The value of N in this file may be calculated using $N_{4\_nodes} = N_{ten\_sec} \times 0.7 \times \sqrt[3]{compute\_cores}$. The 0.7 factor compensates for the fact that no inter-node communication is used during the determination of N_{ten_sec}.
4. Optimize HPL configuration for a benchmark on the small group of nodes. In this step, `runacbea` runs an EA on five populations of forty individuals each for twenty generations. Each individual is evaluated in around ten seconds so this step may take half a day if a single group of nodes is used. The evaluations may be done in parallel over several groups to reduce the time required.
5. Use the best parameters found in step 4 for a new optimization run on groups of nodes four times larger (*i.e* sixteen if step 4 used four), solving problems of double the size: $N_{n^i\_nodes} = 2^{i-1} N_{n\_nodes} \forall i \geq 2$. Repeat this step until you reach a solution suitable for node groups having a size as near as possible to (but not exceeding) the number of nodes in the cluster.
6. Use the best configuration found at the previous step for the final benchmark evaluation on the full cluster. The problem size for this run can be calculated from the cluster's installed memory with the following formula:

   $$N_{full\_theoretical} \simeq 0.8 \sqrt{\text{Total Memory Size in bytes} \times \frac{\texttt{sizeof(double)}}{8}}$$

   The perfect value of N should be manually adapted from $N_{full\_theoretical}$ by monitoring the memory usage on the cluster nodes to avoid swapping. This is an activity that ACBEA does not currently automate. Each run of this last step takes one hour on a cluster having up to 500 cores and 1–2 GiB of memory per core. Note that it is the only step that requires full cluster reservation.
7. Choose the best result for publication as the HPL benchmark score.

4 Scalability

The methodology implemented by ACBEA is based on two assumptions: (1) a single run of an experiment will produce a result that is representative of the

results of multiple runs of the same experiment and (2) HPL parameters that produce good results in small, short benchmarks are likely also to produce good results in larger, longer tests. If the first assumption is not true, the fitness values used by the EA may not be correct, with the result that the next generation does not reflect the genomes of the truly most fit individuals. This issue is investigated in §4.1. If the second assumption is false, there is no point in trying to use small benchmarks to explore HPL's parameter space; large, long-running tests would be the only ones that could yield useful information about full-cluster benchmarks. §4.2 reports on tests of scalability.

4.1 Individual Benchmark Repeatability

A series of tests was run on fifteen two-core nodes of the Chaos cluster (see table 2) to investigate the variability in the results obtained from repeated runs of the same test. As figure 3 demonstrates, variance expressed as a percentage of the result value drops rapidly at first, but the improvement becomes slower as run time increases. This suggests that with this configuration, an N chosen to give a run time of approximately twenty seconds provides a reasonable compromise between the duration of an ACBEA run (which typically entails 4,000 individual benchmarks) and the expectation that a single result is representative (better interconnect than Gigabit Ethernet was found to reduce variability). In further tests (not reported here), variability reduced (and, of course, execution time increased) as the number of nodes assigned to the problem was reduced. Consequently, an execution time of ten seconds is sufficient for benchmarks involving a small number of nodes.

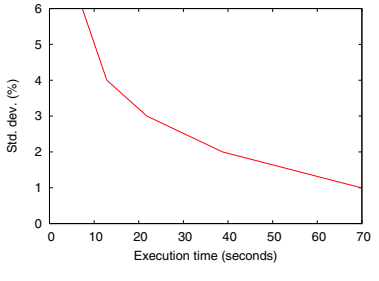


Fig. 3. Variance in results of repeated tests

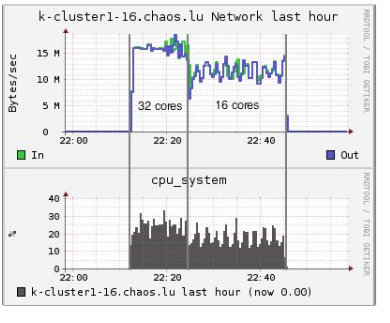


Fig. 4. Effect of a badly-sized test

An alternative way of interpreting the findings is that the problem must not be too small for the number of cores allocated to solve it. If it is, communications activity begins to dominate calculation, resulting in performance figures that are both poor and highly variable. This is illustrated in figure 4, which shows the system CPU time used by a dual-core cluster node involved in solving the same problem ten times, first on 32 cores, then on sixteen. In the 32 core case on the

left, the percentage of system time is higher, indicating that the problem is badly sized for the larger number of cores.

4.2 Interdependence between Parameters

Our earlier paper [6] reported an investigation into the effect of N, problem size, on the optimum value of NB, block size, a parameter found to have a large effect on performance. The conclusion was that the two were independent, with the result that small problems could be used to determine an optimum value of NB that would also be valid for large problems. The work did not investigate the scalability of other parameter combinations, nor did it check whether the findings were specific to the Intel platform, or to the Linux libraries and tools used. Further studies reported here address these issues, and broadly confirm that the results of small benchmarks may be used as a basis for larger experiments.

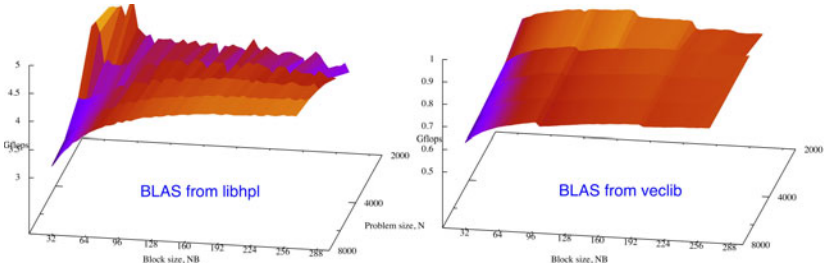


Fig. 5. NB versus N on HP Precision Architecture

N versus NB. In order to check whether the earlier conclusion was true in general, similar tests were run on other platforms and with a variety of BLAS implementations. Space precludes reporting these in detail, but they confirmed the original findings. Figure 5 shows representative results obtained with two different BLAS libraries on a four-core HP/PA host running HP/UX.

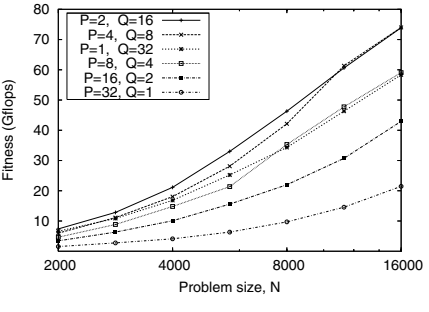


Fig. 6. Relative performance of matrix shapes versus N

P, Q versus N. To divide work among a number of compute nodes, HPL configures the nodes into a $P \times Q$ matrix. The shape of the matrix affects communications patterns and volumes between particular pairs of nodes. An investigation was carried out into whether a shape that was optimal for small problems was also optimal for large. Figure 6 shows a sample of the results. Increasingly large problems are solved while P and Q

are varied, keeping their product, and HPL's other parameters constant. It can be seen that the ordering of the curves barely changes as N is increased, suggesting that information gained from small problems about matrix shape can be applied in large problems. Because adjacent curves do cross on occasions, ACBEA includes new dimensions that are related to the old when creating the configuration file for a subsequent run.

SWAPPING versus N. Studies were also carried out on several machines into the scalability of the $SWAPPING$ parameter, which determines when HPL switches from one data-exchange strategy to another, and which has been observed to have much less effect on benchmark performance than NB or $P \times Q$. Again, the trials suggested that a $SWAPPING$ value that produces good results in small benchmarks will also produce good results in large.

5 Cluster Benchmarking

This section is concerned exclusively with the results of the ACBEA package's automatic tuning of HPL parameters; while it would be instructive to compare automatically-produced results with those obtained by other methods such hand-tuning, or the spreadsheet-assisted procedure proposed in [5], any such study must be the subject of future work. Table 2 describes the clusters that were targeted and the results achieved. It has been remarked, for example in [14,16], that HPL is a good tool for "shaking down" compute clusters. This was certainly found to be the case when ACBEA was built and run on a variety of hosts. Consequently, we are able to report fewer final results here than might have been hoped. More complete descriptions of the French systems that participate Grid'5000 may be found in [1].

Table 2. ACBEA target systems

Cluster name	Location	CPU type/ speed (Ghz)	Total cores	Mem/ core	Inter-connect	MPI	Gflops/ cores
capricorne	Lyon	Opteron/2	112	1GiB	1GE, Myri-2000	MPICH	48/32
chaos-b	Luxembourg	Xeon/3.4	16	4GiB	1GE	OpenMPI	55/16
chaos-k	Luxembourg	Pentium D/3.2	32	2GiB	1GE	OpenMPI	98/30
chinquint	Lille	Xeon/2.8	368	1GiB	Myri-10G	OpenMPI	160/32
genepi	Grenoble	Xeon/2.5	272	1GiB	1GE	MPICH	45/8
granduc	Luxembourg	Xeon/2	176	2GiB	1GE	OpenMPI	671/168
violette	Toulouse	Opteron/2.2	114	1GiB	1GE	OpenMPI	262/96

Chaos-b, Luxembourg. `Chaos-b` consists of just two eight-core nodes. The full ACBEA procedure was run, and a benchmark score of 55.05 Gflops was obtained with $N = 25,600$, $P = 1$, $Q = 16$. This is a considerable improvement upon the disappointing 26 Gflops reported for the same cluster in [6]. The reason for this discrepancy is not known, although the current tests used a better-optimized BLAS library. A study was also made of the repeatability

of the ACBEA process: do repeated runs produce similar or identical recommendations for optimum parameters? The results of four trials of the first optimization phase were in broad agreement. For example, two of the trials gave 72, 96, and 104 as the allowed values for NB in the second phase. (The others gave just 72 and 96, and 72, 104 and 144 respectively.) Other parameter choices were also similar or identical across the four runs. This suggests that the ACBEA process is repeatable — although see the discussion of problem sizing in 4.1.

Chaos-k, Luxembourg. This sixteen-node cluster of two-core nodes was extensively benchmarked for [6], attaining 116 Gflops. One of its nodes was unavailable during the testing reported here. Also, a new and larger Linux kernel made it impossible to use the $N = 84,000$ value used in those tests. Consequently, results are not comparable. After a full run of ACBEA, the five resulting `xhpl` configuration files were used to obtain a best result of 98.47 Gflops with $N = 80,000$, $NB = 88$, $P = 3$ and $Q = 10$. The parameters were derived from those of the fifth-most-successful individual in the optimization run, suggesting that the "best-of-best" individual does not always deliver parameters that are optimum in a larger benchmark.

Granduc, Luxembourg. Currently the largest of the University of Luxembourg's clusters, `granduc` was able to run the full ACBEA procedure. One node being off-line, the final benchmarks were run on 21 nodes (168 cores), giving a best result of 671 Gflops with $N = 192,000$ (using almost all available memory), $NB = 112$, $P = 2$ and $Q = 84$.

Capricorne, Lyon. The `Capricorne` cluster is used by Grid'5000 for experimental work, and was targeted as a test of ACBEA portability because it differs in three respects from the Luxembourg clusters: AMD rather than Intel processors; MPICH instead of OpenMPI; and Myriad high-speed interconnect in addition to gigabit Ethernet. Unfortunately, we were unable to configure MPICH to use the Myriad for data transport, so fell back to using the slower, higher-latency Ethernet. Poor figures were obtained from an initial ACBEA run using eight cores on four compute nodes: the best-performing individual benchmark reached 15.33 Gflops. A second run targeting 32 cores on sixteen nodes obtained a best result of 44.84 Gflops. Because of these disappointing figures, a final test utilizing all cores was not run; the reason for the poor performance was investigated instead. The cause of the problem was found to be incorrect allocation of processes to nodes by MPICH: some nodes were over-subscribed, some under-, and some had the correct number of processes. The reason for this behaviour could not be determined, and the benchmarking attempt was abandoned.

Chinqchint, Lille. A recently-commissioned and powerful system having 368 cores on 46 nodes with ten gigabit Myriad interconnect, `chinquint` proved too unreliable to obtain anything approaching a full-system benchmark. It was possible to run two parallel four-node (32 core) tests for `runacbea`'s full twenty generations. The best individual test delivered an impressive benchmark result of 160.50 Gflops. This was almost twice the overall average of 83.11 Gflops in the final generation. Such a discrepancy is unusual. Sadly, it

was not possible to find sixteen nodes reliable enough to run the next stage of the test, since it should have been possible to obtain well over a teraflop from the whole cluster.

Genepi, Grenoble. Like `capricorne` (see above), `genepi` has MPICH installed on its compute nodes. A first run of ACBEA targeting the eight cores and using eight parallel jobs yielded an average performance of 41.65 Gflops, with the best individual benchmark achieving 44.78. By confining benchmarks to single nodes, this configuration made essentially no use of the interconnect. Sadly, several attempts to run the next stage of the ACBEA process on 32 cores failed to run to completion due to intermittent MPICH problems with secure login between nodes. The experiment was consequently abandoned.

Violette, Toulouse. It was possible to run the complete ACBEA process on `violette` using its installed OpenMPI package. Both stages of optimization performed as expected, delivering five `xhpl` configuration files for final benchmarking. As the cluster has 114 cores (of which some were unavailable) rather than the 64 targeted by the configuration files, the P and Q parameters were adjusted to address 96 cores before final benchmarks were run using $N = 97,600$, a value that was found almost to saturate the nodes' memory. A peak score of 262.3 Gflops was obtained from sixty evaluations derived from the parameters of the five best-performing individuals in the second-stage optimization. As expected, the best result was obtained using the parameters of the "best-of-best" individual. Surprisingly, it used a layout of $P = 16$, $Q = 6$, although over-square matrices generally perform poorly.

6 Conclusions and Future Work

This paper has described how an evolutionary algorithm may be used to produce competitive HPL benchmark results for a computing cluster without the need for intimate knowledge of the benchmark program, or of the software needed to support it. The ACBEA package has proved to be portable to a number of systems, although these have been fairly uniform in operating environment, batch job management and so on. However, portability alone is not sufficient: the target system must be sufficiently robust to support both the demanding benchmark and an evolutionary harness that launches it many thousands of times during the course of an evaluation. At the current state of development, ACBEA still requires a fair amount of knowledge on the part of its user. Files must be edited by hand to set up a starting problem size, and to define the node topology to be used for the evolutionary process. This done, the user must step through the lengthy procedure described in §3.3 in order to obtain a benchmark result. Future work will be focused on increasing ACBEA's ease of use, and on using discovery techniques to reduce the amount of information that must be supplied before a benchmark can be run.

The focus of this paper has been on obtaining results: no attempt has been made to compare ACBEA's results with figures that have been independently obtained by hand-tuning or other methods, either in terms of performance attained,

or of wall-clock time elapsed. It would be instructive to make such comparisons in the future. The work reported in §4 suggests that HPL's other parameters are largely orthogonal to N, the problem size, but does not suggest theoretical or physical reasons as to why this might be the case. Also, all clusters tested to date have provided a fully-interconnected communications topology, which strongly favours a $BCAST$ parameter of zero. Consequently, no information has been obtained as to whether $BCAST$ is scalable or not. Future work could address both of these issues.

The authors would like to thank the administrators and support staff of the Grid'5000 project for their assistance.

References

1. The Grid'5000 Project, http://www.grid5000.fr
2. The Top500 project, http://www.top500.org
3. Cahon, S., Melab, N., Talbi, E.-G.: ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. J. of Heuristics 10(4), 357–380 (2004)
4. Capit, N., et al.: A batch scheduler with high level components. In: Cluster computing and Grid 2005, CCGrid 2005 (2005)
5. Microsoft Corporation. Building and Measuring the Performance of Windows HPC Server 2008-Based Clusters for TOP500 Runs. Technical report (November 2008)
6. Dunlop, D., Varrette, S., Bouvry, P.: On the Use of a Genetic Algorithm in High Performance Computer Benchmark Tuning. In: IEEE International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS 2008), Edinburgh, UK, pp. 105–113 (June 2008)
7. Eigenmann, R., Gaertner, G., Jones, W., Saito, H., Whitney, B.: SPEC hpc2002: The next high-performance computer benchmark. In: ISHPC, pp. 7–10 (2002)
8. Gabriel, E., et al.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Proceedings of 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 2004, pp. 97–104 (2004)
9. Gagné, C., Parizeau, M.: Genericity in evolutionary computation software tools: Principles and case study. Intl. J. on Artificial Intelligence Tools 15(2), 173–194 (2006)
10. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning, January 1989. Addison-Wesley Professional, Reading (1989)
11. Keijzer, M., Merelo, J.J., Romero, G., Schoenauer, M.: Evolving objects: A general purpose evolutionary computation library. In: 5th European Conference on Artificial Evolution, London, UK, pp. 231–244. Springer, Heidelberg (2002)
12. Ladd, S.R.: Acovea: Using Natural Selection to Investigate Software Complexities (2007), http://www.coyotegulch.com/products/acovea/
13. Ladd, S.R.: Evocosm: A C++ Framework for Evolutionary Computing (2007), http://www.coyotegulch.com/products/libevocosm/
14. Levesque, J.: Breakthrough Science on a Petaflop XT5. In: Cray XT Workshop (2009)
15. Levine, D.: Users Guide to the PGAPack Parallel Genetic Algorithm Library (1996), ftp://info.mcs.anl.gov/pub/tech_reports/reports/ANL9518.ps.Z
16. Minyard, T., et al.: Experiences and Achievements in Deploying Ranger, The First NSF "Path to Petascale" System. In: TeraGrid 2008 (June 2008)

17. Petitet, A., Whaley, R.C., Dongarra, J., Cleary, A.: HPL — A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers (January 2004), www.netlib.org/benchmark/hpl/
18. Sastry, K.: Single and Multiobjective Genetic Algorithm Toolbox in C++ (2007), http://www.illigal.uiuc.edu/pub/papers/IlliGALs/2007016.pdf
19. Sripathi, V., Krishnan, A.: Analyze and optimize the HPL benchmark on x86-64 cluster. Technical report, North Carolina State University (2008)
20. Wall, M.: GAlib — A C++ Library of Genetic Algorithm Components
21. Whaley, R.C., Petitet, A., Dongarra, J.: Automated empirical optimizations of software and the ATLAS project. Parallel Computing 27(1-2), 3–35 (2001)

Monitoring of SLA Parameters within VO for the SOA Paradigm

Wlodzimierz Funika, Bartosz Kryza, Renata Slota, Jacek Kitowski, Kornel Skalkowski, Jakub Sendor, and Dariusz Krol

Institute of Computer Science, AGH-UST,
al. Mickiewicza 30, 30-059 Krakow, Poland
{bkryza,funika,rena,kito}@agh.edu.pl

Abstract. Current trends in modern scientific and business IT infrastructures pose certain requirements in the middleware layer in order to maximize the automation of all life-cycle phases of such infrastructures including inception, deployment, execution, and dissolution. In case this infrastructure is composed of resources of different organizations, for instance in form of a Virtual Organization, the management of these resources is especially needed for achieving new quality in business. In this paper we deal with a specific aspect of the IT infrastructure management related to autonomous enforcement of Service Level Agreement between organizations sharing their resources within a Virtual Organization. The presented framework utilizes semantic technologies in order to virtualize the heterogeneity of underlying middleware components and to allow integration of services between these organizations.

1 Introduction

Modern applications of various technologies developed in recent years such as Grid computing or Service Oriented Architectures currently are being adopted in more and more areas of computing including not only research institutions and large corporations, but also more commonly even smaller SME companies. These technologies are advocated as solutions to the problem of integration of resources and services between both large parties as well as small entities. The latter, however, often do not have sufficient funds or know-how in order to transfer their IT infrastructures to modern technologies and then efficiently manage them - an effort which still requires very specialized knowledge and experience.

That is why a crucial element in making use of these widespread technologies is in making the process of participating in such IT based business collaborations much easier and more affordable for even smaller parties. This can be achieved by automating as much of the process of managing this infrastructure within the middleware layer itself and thus limiting the burden imposed on the administrators and the IT staff of these organizations. For this purpose our recent work involves the development of a complex solution for automatic Virtual Organization management supporting several aspects. These include the reaching of an

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 115–124, 2010.

agreement between the organizations in the form of a distributed contract negotiation, automatic configuration, and deployment of the Virtual Organization on the resources and services of organizations participating in such VO as well as autonomous enforcement of the contract through monitoring of execution of the Virtual Organization and proper security configuration.

In this paper we focus on one aspect of the VO management mentioned above that concerns the monitoring and enforcement SLA parameters for the SOA paradigm. This research is part of the work on the FiVO[1] system [1].

The paper is organized as follows: in Section 2 related work on SLA enforcement tools as well as some existing monitoring tools are presented. In Section 3 the architecture of the FiVO SLA Enforcement Tool is discussed in the context of Virtual Organization management. Section 4 gives an overview of the SemMon semantic monitoring tool and, finally, Section 5, describing our approach to integrate FiVO with the SemMon system, is followed by Conclusions.

2 Related Work

The monitoring of SLA fulfilment for VO requires taking into consideration a few aspects: heterogenous environment (different monitoring systems installed in various physical organizations), dynamic changes of SLA requirements, and mapping of high-level SLA parameters onto low-level resources parameters.

The way these three aspects are considered depends on a chosen approach. A first approach to the monitoring of SLA fulfilment is presented in [2]. This approach assumes that everything what is monitored is a Web Service. SLA in this approach is defined using the Web Service Level Agreement (WSLA) framework. WSLA is an IBM framework which supports SLA management. High-level SLA parameters are mapped onto low-level resources parameters which are described by the Common Information Model (CIM). In [3] there is presented an approach assuming that the grid environments which follow the Open Grid Services Architecture (OGSA) are monitored. The monitoring of SLA fulfilment on grids and mobile grids are reached by introducing the Execution Management Service which monitors both the network services and the application services. Another approach is presented in [4]. In case of this approach only the network services are monitored. QoS attributes are achieved in this case by implementing special services. All the above approaches described are oriented toward monitoring SLAs in special environments (e.g. grids). A comparison of some existing architectures oriented towards SLA monitoring presented in [5] shows that there is no single architecture which could be applied in every case. In case of the approach presented in this paper we focus on the issue of independence from the low-level monitoring frameworks, which is a key feature if one wants to use the FiVO framework in different physical organizations. This approach is needed in case of the VO management when we deal with a heterogenous environment, consisting of different low-level monitoring systems used in different physical organizations.

[1] Framework for Intelligent Virtual Organizations.

A very important part of every SLA management framework is a monitoring system, so below we present some of the existing monitoring systems.

The PushToTest TestMaker system [6] is a testing platform, which provides support for various kinds of tests (functional tests, load tests, stress tests, performance tests, smoke tests and others). It also provides support for enforcing SLA. All tests are defined in a manually written script file.

Monitoring & Discovery System (MDS) [7] from Globus Toolkit Information Services allows to discover resources (considered as a part of a virtual organization), which are available on the grid and to monitor these resources. It provides two services which determine the interface of this system: the index service and the trigger service. The index service provides a query interface which allows to send queries about monitored parameters. The trigger service allows to configure special actions based on the monitoring data. The data provided by this system contain the data on CPU and memory usage, size of queues, etc.A part of the WS-Diamond (Web-Service Diagnosability, Monitoring and Diagnosis) system [8] is a monitoring framework for QoS attributes, which enables to detect a failure situations, and also permits to plan and perform different recovery actions. This functionality helps to achieve QoS objectives.

Grid-enabled OMIS-Compliant Monitoring System (OCM-G) [9] is oriented towards support for performance analysis tools, for evaluating parallel and distributed applications, especially, those using MPI, and for monitoring grid resources. Java Oriented Monitoring Infrastructure J-OCM [10] is aimed to monitor and handle the execution of Java distributed applications. Like the OCM-G system it is also oriented towards support for performance analysis tools.

Since the above systems are oriented towards the monitoring of resource parameters, they should be treated as low-level monitoring systems. In case of monitoring for VO we need a system which enables automatic monitoring of SLA fulfilment, based on a contract content. None of the above systems provides it. The PushToTest TestMaker system [6] allows to define complex tests, but it is done manually, by writing a special script, so it cannot be useful in our case. The other systems mentioned above are typical monitoring systems, which are not targeted to monitor SLA fulfilment, sometimes they help us to achieve QoS objectives only (e.g. WS-Diamond).

In case of the monitoring of SLA fulfilment in a heterogenous environment, comprised of different low-level monitoring systems, we need more sophisticated functionality which should constitute a bridge between the resources abstraction and the SLA abstraction, e.g. in form of a medium-level monitoring system, which is able to map composite SLA metrics into low-level resources parameters. This system has to provide a kind of semantic-reasoning engine, which allows to map high-level SLA parameters onto the low-level resource parameters. One of such systems is the Semantic Monitoring System (SemMon) [11]. A key feature of the SemMon system is its independence from a low-level monitoring system, what means that the SemMon system constitutes an abstraction layer over heterogeneous low-level monitoring systems. A more detailed description of the SemMon system appears in Section 4.

3 SLA Enforcement in Virtual Organizations

In general, FiVO, as mentioned in Introduction, is a comprehensive and distributed framework allowing for management of all the aspects of VO life-cycle. Once an organization deploys its components in its premises, it becomes part of a Virtual Breeding Environment, i.e. it can become a member of existing and emerging Virtual Organizations managed by FiVO. In order to support this, FiVO provides the organizations with several components responsible for the distributed contract negotiation phase [12] as well as components providing security [1] and SLA enforcement in a VO context as defined in the contract. In particular, in order to support the SLA enforcement aspect of VO, each organization must deploy the FiVO SLA Enforcement Tool and allow access to its legacy monitoring infrastructure. After the contract is negotiated for a new VO, the contract statements specific to each organization will be deployed by proper FiVO instances in the underlying monitoring systems.

An example VO monitoring environment is shown in Fig. 1, where it is assumed that inside a real-life organization all resources are monitored by a single high-level monitoring system which gathers information from one or more low-level monitoring systems. The environment of monitoring systems across different real-life organizations in a Virtual Organization is usually heterogeneous. Different organizations can monitor their resources using different systems and even inside an organization, multiple types of monitoring tools may be used for different types of resources. The monitoring of contract fulfilment and SLA enforcement procedure forces to build a VO monitoring tool which can cope with the following requirements:

- interpreting of contract statements specified in the form of OWL ontology,
- automatic configuration of the legacy monitoring tools used in organizations participating in a new VO,

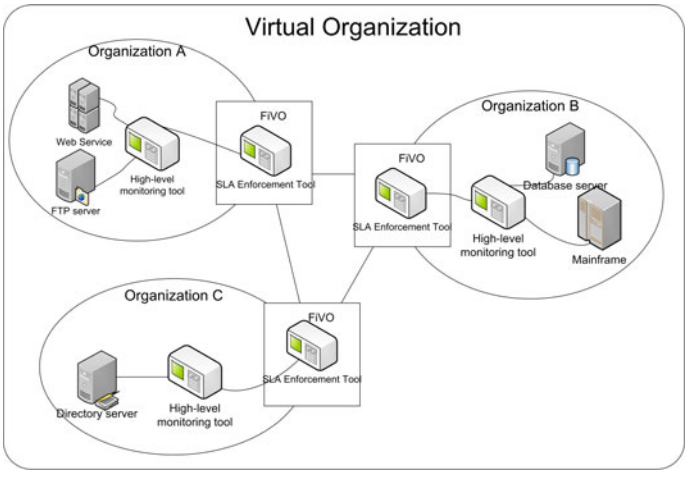


Fig. 1. Virtual Organization monitoring environment

– distributed architecture which does not require that the organizations provide access to their middleware for external administrators
– ability of triggering corrective actions if contract statements are violated.

In FiVO SLA Enforcement Tool we prepared an architecture design consisting of several modules which allow FiVO to meet the above requirements (please see Fig. 2 - for simplicity, we assume that 3 organizations are sharing one FiVO SLA Enforcement Tool instance). The values of SLA parameters are analyzed and extracted by the **Ontology Analyzer** from a VO contract. Contract statements are negotiated between real-life organizations and are described using an OWL ontology. As a result from Ontology Analyzer we acquire a list of the expected QoS parameter values, retrieved from the SLA part of the contract, for each resource or service shared by a real-life organization inside the VO. In the next

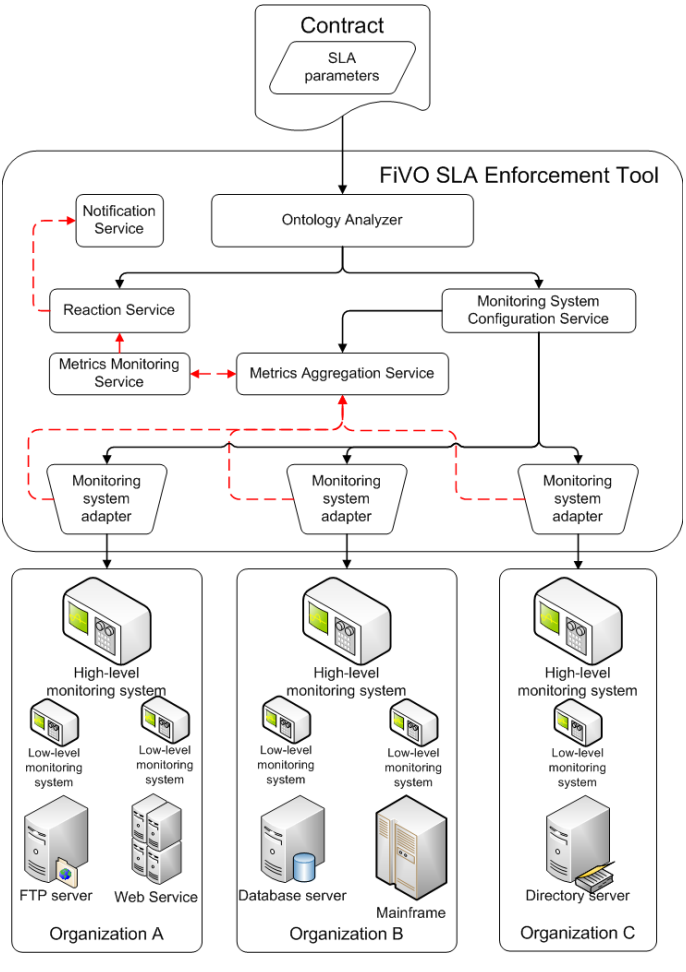


Fig. 2. SLA monitoring tool architecture overview

step, performed by **Monitoring System Configuration Service**, QoS parameters are grouped by real-life organizations. Sets of metrics and measurements are prepared for each organization which may use any high-level monitoring system. The configuration of a specific monitoring system is done by adapters, used to provide an abstraction layer between an organization-specific system and FiVO SLA Enforcement Tool. A set of measurements for a single real-life organization resources is passed to the adapter responsible for configuring the organization-specific monitoring system to prepare measurements and retrieving metrics values from it.

Along with the configuration of measurements goes the set-up of the notification system part. Penalty clauses or declared sets of actions to perform in the situation when SLA statements are not met can also be described within the contract. These actions are passed to **Reaction Service**. We also introduce two more services that are required in order to provide the monitoring of SLA parameters during the VO life-cycle:

- **Metrics Aggregation Service** - which enables storing metrics values history and provides access to aggregated values (e.g. mean metric value from one day, one month)
- **Metrics Monitoring Service** - which fills the gap between the Reaction Service and monitoring systems; major role of this service is monitoring metrics values or aggregates values and triggering actions in the Reaction Service.

High-level monitoring systems are usually capable of doing metrics aggregation or statistics so the complexity of processing done by these services depends on the real-life organization monitoring system features. The Reaction Service starts receiving notifications when the expected SLA parameters values are not met. This information can be passed further to different information channels (e-mail, instant messenger, published on RSS/Atom feed) via the **Notification Service** or a relevant action can be invoked by RMI or Web Services.

The proposed solution can also handle situations when we collect information directly from low-level monitoring systems. In this case, communication between an organization-specific monitoring tool and its adapter is increased. Also the amount of data stored and processed by the Metrics Aggregation Service is higher. Therefore using a high-level monitoring tool (e.g. SemMon, which is capable of aggregating metrics from one or more low-level monitoring systems) on the real-life organization site is very helpful. SemMon's ability to hide the complexity of low-level monitoring tools inside the real-life organization enables to reduce difficulties connected with creating a custom metrics aggregation and monitoring service. It should also decrease the amount of time spent on the development of an adapter compared to the work on low-level monitoring tools.

4 The SemMon System Description

The SemMon[2] monitoring system [11] is placed on top of the existing monitoring stack and provides an additional functionality which is not possible to achieve

[2] SemMon stands for *Sem*antic-based *Mon*itoring system.

otherwise. It aims to map data from the underlying low-level monitoring systems, intended to provide *physical* information about the monitored system health onto high-level information. By saying *high-level information* we mean performance metrics that are not provided by the monitoring systems but can be derived from them, e.g. as a combination. The SemMon can also notify the interested users about previously defined situations, e.g. when a measurement of a given metric and resource passed a defined threshold. This functionality may be helpful when one wants to be informed about the violation of the previously defined conditions. To do so a *knowledgebase* that contains semantic information about available resources and metrics is used. It is described more detailed in the next subsection.

An important feature of the SemMon system is its capability to cooperate with any existing low-level monitoring system that exposes its data to the external clients. The cooperation takes place through the commonly used *adapter* structural design pattern [13] and it is used for 'translation' of all requests from the internal SemMon format to the format appropriate for the underlying monitoring system, e.g. J-OCM [10], OCM-G [9], etc.

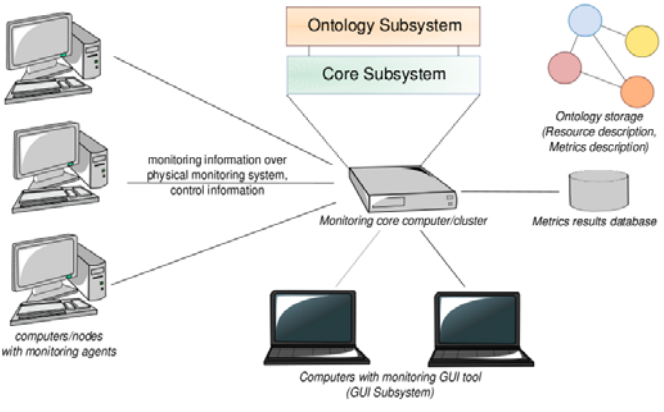


Fig. 3. The SemMon architecture overview

4.1 System Architecture

An overview of SemMon's architecture is depicted in Fig 3. The presented system along with the monitored objects constitutes a kind of distributed system. There can be multiple nodes with monitoring agents and multiple nodes with GUI. The *heart* of the system is also separated into different subsystems (and components), each of them can be run on a separate machine to provide scalability and reliability. Each subsystem encapsulates some specific functionality and can be run independently. The main elements of the SemMon monitoring system are as follows: Core and Ontology subsystems, Monitoring agents, and Graphical User Interface.

The key part of the system are Core and Ontology subsystems that provide primary system functionality like processing an ontology with Resources and Metrics, or storing monitoring data. To support knowledge persistency, a database is required. This functionality is implemented in an ontology subsystem. Another part of this node is the support for a 'physical' monitoring system. This subsystem has to provide functionality for registering *monitoring agents* as well as processing the monitoring data.

Monitoring agents are just simple sensors that expose resources to the SemMon system in a well-defined way. All of the agents have to be registered in the core part of the SemMon. Afterwards the Core can ask about available resources and capabilities that can be monitored. High-level monitoring data is available to external clients through an interface exposed by the Core subsystem. An example of such external client is GUI which can attach to SemMon using the Remote Graphical Interface. Similarly, other clients handling high-level information, like SLA parameters or their derivatives can use the interface exposed for using SemMon functionalities.

All of the core system functionality should be accessible via GUI (e.g. browse a monitored resource, run a metric) in a simple and user friendly way. It has to be optimally designed for the advanced user as well as for the beginner. It is achieved on the one hand by exposing the most often performed options to the foreground and on the other hand by allowing users to provide detailed configurations options to the available operations. GUI will provide functionality for collecting some parts of the knowledge base data - like a metric rank. GUI is also an environment for collaborative work, e.g. users could share metrics ranks between their instances of GUI.

4.2 Knowledge Usage

SemMon is a semantic-based monitoring system which means that it has some inbuilt semantic knowledge about metrics and monitored system resources, which can be explicitly extended by the users. It enables to perform monitoring at different levels, from detection of hardware errors to high-level analysis. In this context, semantic knowledge can greatly contribute to a multi-layer and multi-source monitoring process giving the end-user an easy and efficient tool for semi-automatic data analysis and monitoring guidelines. One of the possible benefits that the user may get from the SemMon is providing suggestions on performing an additional measurement of a metric semantically-related to the one that is currently running. By doing so the user may explore the actual problem (e.g. bottleneck) of the monitored application faster. It is especially helpful when the user does not have thorough knowledge about the application and the underlying infrastructure used. Another important aspect of using the ontology knowledge representation which should be mentioned is the ability of customization to specific situations. It is easier to provide support for new requirements (and therefore to add some code to the core about the new classes of resources) than to rewrite the whole business logic of the monitoring system which is, in most cases, oriented to some particular type of applications.

5 SLA Support for VO

SLA contracts play a very important role in case of the FiVO system. During the negotiation process, sides are agreeing what resources and on what terms are to be available in the organization. The most important part of SLA from the monitoring point of view is the one that holds information about the QoS aspects. The ontological form of an SLA contract may be used to integrate the systems that are involved in the management of Virtual Organizations. On the FiVO system side, the knowledge about an agreement can be used to inform the monitoring system about the monitored resources and metrics which should be measured. Furthermore it can be used to define the conditions upon which FiVO system should send a notification about a contract violation. Thus, the FiVO system can be used as a guard that checks if the Service Level Agreement is fulfilled properly. In addition, it can notify the management tools or the system administrator about the contract violation. By applying the described functionality the Virtual Organizations can become more reliable and efficient in achieving their goals. By using a high-level monitoring system to monitor particular physical organizations we reach full independence from low-level specific monitoring systems and improve the scalability and robustness of the FiVO system. The scalability improvement results from dispatching the process of mapping composite SLA parameters to low-level resource indicators from the core of the FiVO system to the SemMon system engine.

6 Conclusions

In this paper we have presented a solution to the issue of monitoring and enforcing agreements between partners in Virtual Organizations. The FiVO framework for management of Virtual Organizations was presented along with its integration with the SemMon monitoring system. The proposed solution can improve the adoption of modern Grid or SOA based infrastructures in various environments, especially, where the main problem is the lack of sufficient know-how required to deploy and manage complex middleware. Additionally, the automation of the management process allows to cut down on the costs of monitoring of the cooperation of organizations based e.g. on SOA frameworks and thus increase the level of collaboration between organizations based solely on the IT infrastructures.

Acknowledgments. This research is partly funded by the POIG.01.03.01–00-008/08 Project "IT-SOA" and the AGH grant 11.11.120.777.

References

1. Kryza, B., Dutka, L., Slota, R., Kitowski, J.: Security Focused Dynamic Virtual Organizations in the Grid based on Contracts. In: Cunningham, P., Cunningham, M. (eds.) Collaboration and the Knowledge Economy, Issues, Applications, Case Studies, part II, vol. 5, pp. 1153–1160. IOS Press, Amsterdam (2008)

2. Debusmann, M., Keller, A.: SLA-Driven Management of Distributed Systems Using the Common Information Model. Integrated Network Management 246, 563–576 (2003)
3. Litke, A., Konstanteli, K., Andronikou, V., Chatzis, S., Varvarigou, T.: Managing service level agreement contracts in OGSA-based Grids. Future Generation Computer Systems 24(4), 245–258 (2008)
4. Bouras, C., Campanella, M., Przybylski, M., Sevasti, A.: QoS and SLA aspects across multiple management domains: the SEQUIN approach. Future Generation Computer Systems 19(2), 313–326 (2003)
5. Barbosa, A.C., Sauve, J., Cirne, W., Carelli, M.: Evaluating architectures for independently auditing service level agreements. Future Generation Computer Systems 22(7), 721–731 (2006)
6. PushToTest TestMaker project site: http://www.pushtotest.com/products
7. Globus Toolkit Information Services: Monitoring & Discovery System project page, http://www.globus.org/toolkit/mds/
8. WS-Diamond (Web-Service Diagnosability, Monitoring and Diagnosis) project page, http://wsdiamond.di.unito.it/
9. Balis, B., et al.: Grid Environment for On-line Application Monitoring and Performance Analysis. Scientific Pogrammning 12(4), 239–251 (2004)
10. Funika, W., Koch, M., Dziok, D., Smetek, M., Wismüller, R.: Performance Visualization of Web Services Using J-OCM and SCIRun/TAU. In: Yang, L.T., Rana, O.F., Di Martino, B., Dongarra, J. (eds.) HPCC 2005. LNCS, vol. 3726, pp. 666–671. Springer, Heidelberg (2005)
11. Funika, W., Godowski, P., Pęgiel, P.: A semantic-oriented platform for performance monitoring of distributed Java applications. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part III. LNCS, vol. 5103, pp. 233–242. Springer, Heidelberg (2008)
12. Zuzek, M., Talik, M., Swierczynski, T., Wisniewski, C., Kryza, B., Dutka, L., Kitowski, J.: Formal Model for Contract Negotiation in Knowledge-Based Virtual Organizations. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part III. LNCS, vol. 5103, pp. 409–418. Springer, Heidelberg (2008)
13. Johnson, R., Gamma, E., Helm, R., Vlissides, J.: Design Patterns. In: Elements of Reusable Object-Oriented Software Addison-Wesley, Reading (1995)

A Role-Based Approach to Self-healing in Autonomous Monitoring Systems

Włodzimierz Funika and Piotr Pęgiel

Institute of Computer Science, AGH,
ul. Mickiewicza 30, 30-059 Kraków, Poland
Ph.: (+48 12) 617 44 66; Fax:(+48 12) 633 80 54
funika@agh.edu.pl, barca@wp.pl

Abstract. The main intention of this paper is to introduce the proposition of a new role-based approach to self-healing monitoring. This is preceded by an overview of existing approaches to the monitoring of distributed systems using self-healing features. Starting with a discussion of autonomous monitoring systems, we will come to self-healing systems. These systems should be able to automatically resolve the problems that occur in a system under monitoring. The paper provides insight into various aspects of self-healing monitoring systems at the software and hardware level. A detailed description of a new agent-based system, AgeMon, is covered later on. The system is based on the roles played by different types of agents. The self-healing features can be achieved by a form of cooperation of agents, e.g. monitoring agents, rule agents, database agents. The paper discusses the roles and gives an implementation background.

Keywords: Self-healing, monitoring, adaptive, rule-based systems, failure detection.

1 Introduction

Nowadays systems are becoming very complex. They are, in fact, very frequently built with many components which are working on different machines, in a *distributed environment*. It is impossible to monitor such systems manually, there are too many different indicators to check (resource states, network traffic, operated system, etc.). This was the main reason why distributed monitoring systems were developed. They help the user in managing a system – usually the user is enabled to observe all interesting data in the monitoring system presentation layer. In such systems the user is responsible for interpreting encountered problems and perform relevant actions.

The next stage in the evolution of monitoring systems is connected with the concept of autonomous monitoring systems. Such systems do not need user interaction to make a proper decision what should be monitored in a current situation. Decision making is based on the knowledge gathered from the preceding monitoring results. The system could also *guide* the user what should be done next, i.e. in the context of monitoring.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 125–134, 2010.

To fulfill these requirements for the guidance to the user, the monitoring systems became more "intelligent". From this point it was a straight way for coping with arising failures of different types – simply to enable self-healing, i.e. to provide a capability that a decision made by the monitoring system should force the system under monitoring to behave more stable, reliable and predictable.

There can be more than a single aspect of self-healing: the monitoring system could "heal" the system under monitoring or it could be "healed" by itself. We can also distinguish between the levels of healing: the system can perform self-healing in the physical (monitored hardware) layer or the logical (monitored application/system) one. A brief of the monitoring systems evolution is depicted in Fig. 1.

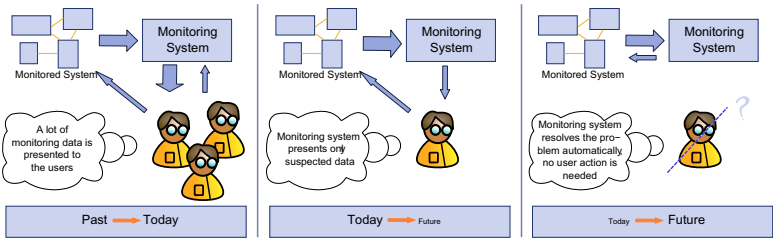


Fig. 1. Evolution of the monitoring systems

In the paper we aim to present an overview of monitoring systems and techniques used in them for self-healing.

The rest of this paper is organised as follows: Section 2 discusses details of self-healing monitoring systems at different levels. A case study of the existing self-healing systems, technologies and approaches is provided in Section 3. The next section introduces requirements for the self-healing monitoring system together with the idea of a new approach to self-healing and a description of a new system – AgeMon, followed by Summary.

2 Self-healing Monitoring Systems – Background

Two main aspects of self-healing monitoring systems can be distinguished between when self-healing systems are concerned. The first aspect is related to the physical layer of the system (like computers, resources, and network), while the second aspect regards the logical layer (applications, operating system). Monitoring the physical layer is usually more intuitive. We can imagine a situation when an operating system can make a decision to automatically offline a faulty resource. This functionality could even be implemented on the system level – like in the Solaris 10 [1].

In the second aspect of self-healing, a needed functionality can be injected into the monitored system. In this situation, technologies like Aspect Oriented

Programming can be used [2,3]. Using AOP techniques we can cross-cut the business logic of the application to inspect its status. When an incorrect state is detected the monitoring system can perform a recovery action. There are also efforts to extend existing monitoring systems to enable self-healing - e.g. the Nagios system can be used to automate recovery after service failures [4].

We can also consider self-healing from a different point of view: In case of the first approach mentioned above – the self-healing of the monitored system, the monitoring system should automatically detect failures in the system under monitoring and *heal* it. The second approach is quite interesting as well – the monitoring system can be able to perform a self-healing action "on itself"– we can imagine that the system is composed of many autonomous agents which can make a decision to disable an unstable agent.

A key point is to answer such questions as: How can we maximise availability, reliability, safety, and maintainability for the monitoring system? How can the monitoring system detect failures in its infrastructure, prevent from their affect, and self-heal? Such a system should be distributed, and use a flexible communication protocol with failure detection. The decentralisation of the monitoring system can be realised within an agent-based network. Some agents can be specialised in one of the following categories: monitoring agents, agents responsible for storing monitoring data to the database, knowledge based failure detection agents, decision agents, UI communication agents. Due to specialisation, one of the election protocols should be considered. Some agents can be replicated like those responsible for storing the data to the database. In case of problems with one agent the system should detect a failure and perform an election procedure to find a new persistent agent.

3 Overview of Existing Self-healing Monitoring Approaches

In this section we aim to present the existing self–healing monitoring systems and technologies that can be used to enable self-healing in systems under monitoring. We will look at Aspect Oriented Programming and the existing solutions that are based on this technology. Then we will discuss Solaris 10 – it is an example where self-healing is done on the system level. At the end we will review a real-life example of self-healing system.

3.1 Application Layer – Aspect Oriented Programming

Aspect Oriented Programming is a programming paradigm which enables separation of concerns (which increases the modularity of the program). It allows also cross–cutting - for the place of the program where concerns "cut across" multiple abstractions in the program. Procedures, functions, modules, classes – these are the abstractions in the program which are used to group the concerns. The encapsulation of concept is a good way of programming - but sometimes it is required that some of the functionalities should "go across" the whole program. This is the *cross-cutting concern*. Currently, the main usage of the AOP

techniques in the self-healing is to enable system recovery [3,6,8]. The decision when a recovery should be performed is based on a user's choice. Therefore these systems cannot be treated as autonomous.

Glassbox. One of the most mature solutions for monitoring using Aspect Oriented approach is the Glassbox Project[1]. *Glassbox* is a Java based monitoring tool with an extended troubleshooting module. It helps developers in resolving common problems like failing connections or a slow-running query instantly. It can be used with most of the Java Application Servers (like JBoss, WebSphere, WebLogic, Tomcat). Since it contains predefined knowledge it can be used just after download. The problems are described in a plain English – without any logs or complicated messages.

From the practical point of view, Glassbox instrumentation could be done at compile time or at class load time (so Glassbox does not require code recompilation). Fig. 2 depicts the structure of the Glassbox [9].

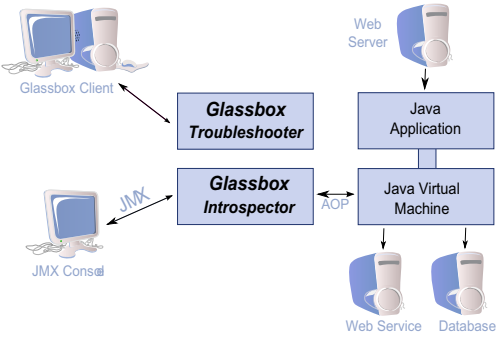


Fig. 2. Glassbox architecture

There are two main subsystems of the GlassBox: Glassbox Inspector and Glassbox Troubleshooter. The Inspector is used to monitor the activity of an application. Monitoring information is exposed via JMX interface so it can simply be read from any Java JMX client like JConsole. On the other hand this information is passed to the Troubleshooter. This module is responsible for making analysis and diagnosing problems and reporting them them trough a specialized client UI.

The Glassbox is a powerful tool which easily enables troubleshooting of running an application with guidance to the user what action should be taken. It does not require changes to the application code. It can also be used as a infrastructure layer for bigger monitoring systems which extend the functionality with autonomous decisions – the system will be able to automatically make a decision without interactions with the user.

[1] Home page – http://www.glassbox.com/glassbox/Home.html

AOP-Monitoring framework. While Glassbox is a good example of the Aspect Oriented Monitoring – it is simple to use but very helpful, it is however not an ideal tool – e.g. it is not possible to change the monitoring level at real time. At the same time there are solutions that aim to resolve this problem, one of them being *AOP-Monitoring framework* [2].

AOP-Monitoring framework is based on a simple architecture presented in Fig. 3 a.

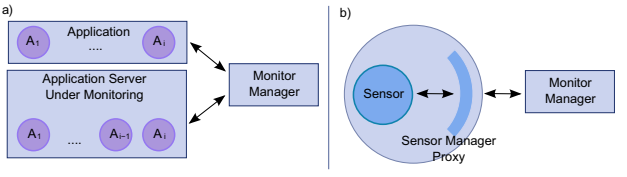


Fig. 3. AOP-Monitoring framework architecture: a) overview, b) sensor details

There are two main parts of the framework: Sensors and Monitor Manager. Sensors are responsible for collecting data from the system - they are in fact *aspects* injected into the code. *Sensors* can be defined on different monitoring levels, e.g. when a method or constructor is called, before object initialisation, before a field is read, etc. Data is purged to the Monitor Manager once a defined threshold is achieved. In Fig. 3 b., there is a detailed presentation of the sensor architecture. It is a composition of a real sensor (an entity defined using AspectJ [10] and a Sensor Manager Proxy. The Sensor Manager Proxy is aimed to communicate with the Monitor Manager.

The *Monitor Manager* is responsible for collecting all the data from all active sensors and determine what policy should be applied. The Monitor Manager allows failure prediction by using data mining or forecasting methods using the collected sensor data. If the prediction methods need more data from the system or low level monitoring, the Monitor Manager can activate new sensors and deactivate others to obtain these data to accurately determine if there will occur a failure or an error.

The AOP-Monitoring framework is a recent approach to the monitoring with aspects. It is flexible and uses well known technologies like AspectJ. Currently there is no final version of this framework – it is under development, there is a first version but without data mining or prediction methods.

4 Self-healing Monitoring System

In this section we present requirements for the self-healing monitoring system together with a case study of the problems that may occur during the prototyping. At the end we are going to present our solution.

4.1 System Architecture

The monitoring of a distributed system is not a trivial task. The system components are usually located on different nodes or machines and communicate

through network. The most common (and probably the most natural) way of monitoring such systems is to use an agent-based approach [7]. In this case the monitoring system usually consists of a set of agents and some other components like database with monitoring results or a User Interface. The agents are used to pull monitoring information from the monitored system nodes. In self-healing approaches such agents pull the information to the *Oracle* [17] which is used for failure detection (described broader in the next subsection). As usually additional components are running on separate machines. The problem with this approach is the fact that a failure of one of these components leads to a failure in the whole monitoring system (when a component with results is down the system is not able to store any results). Therefore before enabling self-healing in a monitored system the monitoring system should be able to heal its internal problems.

4.2 AgeMon – Self-healing Monitoring System

To enable self-healing in the monitoring system, some additional work should be done during the system design. Since the system is going to be based on the agents(it is a reasonable approach in a distributed environment), and all of the system components should work and behave as agents. This approach simplifies the design and deployment of the monitoring system. Of course there should be a kind of specialisation of the agents. In our approach we prefer to use term *roles* that can be fulfilled by the agent. The following roles are designed:

- *Regular monitoring role* – this agent is used to collect the monitoring data from the monitored system (it is working like a sensor). The data can be gathered from various sources – directly from the application (e.g. by AOP), by a specific monitoring system (like JMX), from the operating system (native libraries, statistics files). This agent can also be used to enable self-healing in the monitored system.
- *GUI role* – used to present the monitoring results to the user and allows to manage the monitoring system. The user should be able to observe the current state of the monitoring and monitored system, manage monitoring (start, stop), see the results displayed on-line, compare different monitoring results in one visualisation window.
- *Database role* – used to store the monitoring information in a persistent database. The agent should also be able to serve the historical monitoring information to other agents on demand.
- *Rule role* – the agent used to transform the monitoring results (e.g. by running metrics). Dependent on the results of the transformation the agent should be able to *decide* on what action should be taken after the transformation (if any). There can be many different actions, e.g.: run *a new metric*, send *a predefined event to the monitoring system*, send *a predefined event to the monitored system*, send *a notification* to the user.

It is possible that an agent is able to work in more than one role. We can imagine a situation when the agent that gathers the data, automatically pre-process them and stores to the embedded database along with sending the results to GUI.

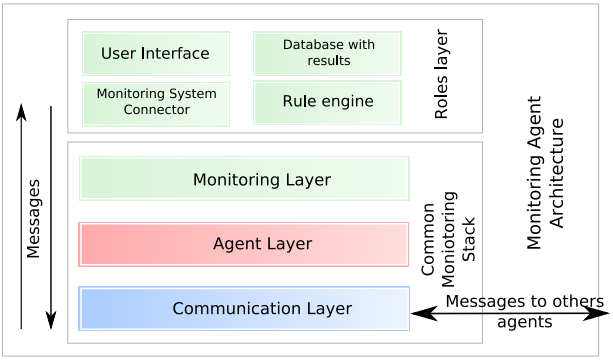


Fig. 4. Stack-based monitoring agent

Due to the above concept – one agent should be able to work in different roles – the architecture of the agent should be based on well defined interfaces that split implementations into well defined layers. Fig 4 presents an agent designed based on this approach.

The main requirements for each layer:

- *Communication Layer* – this layer is used for communication between agents. To save network bandwidth it should allow messaging to more than a single agent at the same time (e.g. when an IP network is considered it should allow IP-multicast). It should support tunnelling for larger networks, and be as user-friendly (*zero-configuration*) as possible. One of the most important requirements for this layer is to provide automatic discovery of a new agent attached to the network.

During the prototype implementation we tested multiple libraries and protocols that can be used as a base in this layer: SLP [11] (*OpenSLP*, *jSLP*), SSDP [12] (Microsoft UPnP SDK, UPNPLib) and JINI [13]. We have decided to use JGroups[2] – it is a lightweight, reliable multicast communication toolkit with discovering of new agents. Of course the implementation is hidden behind the interfaces, therefore it should be easy to change the implementation.

- *Agent layer* – implements the agent abstraction. This layer is a helper layer between the communication layer (where each agent can be distinguished by the name only) and the monitoring layer. It is responsible for keeping the updated agent group that belongs to one monitoring group. It also hides the implementation of remotely executed monitoring tasks.

- *Monitoring layer* – is responsible for processing monitoring messages passed between agents. The messages can be divided into two groups: regular monitoring messages (with results of the monitoring) and control messages. Control messages contain information about actions made by user/system and inform about

[2] `http://www.jgroups.org/`

the current state of the system (e.g. present a list of the running monitoring instances, describe capabilities, start/stop monitoring sessions).

- *Roles layer* – this layer contains the implementation of each role. Due to the layer separation, each role can reuse lower layers independently.

In addition, based on the above idea, some other functionalities should be provided to enable self–healing in the monitoring system. Since the communication layer is responsible for automatic discovery of new agents and notification if an agent fails (disconnects from the network) an election algorithm can be started to find a new agent that can be used for the selected roles.

Let's consider an example with 4 agents - 2 monitoring agents, 1 transformation agent, and 1 database agent. When the database agent fails other agents can start an election to find the agent that can work as a database agent (store the monitoring results) until the primary database agent is restarted. Distributed cache is also a nice technology that can be used to enable self-healing. The monitoring configuration (what is monitored by an agent) can be stored in such a cache and used to restore the monitoring configuration when network failures are detected. The rule agent can be used to enable self-healing of the monitoring system. It can be used for detecting the failures in the monitored system as well as in the monitoring system - e.g. we can define a rule that whenever the database agent is down the system should notify the user with a problem description.

4.3 Enabling Self-healing in the Monitored System

Enabling self-healing in the monitored system is not a simple task. It can be considered with different granularity levels [14,15].The most common solutions are working with coarse-grained components. A common example is related to load-balancing. A monitoring system based on the current load factor can make a decision to run additional nodes that can handle more transactions per second.

More recent self-healing approaches operate at the system's architectural level by exploiting architecture reconfiguration strategies. The ability to dynamically add redundant servers, add or remove components are some examples.

Enabling self-healing on the class or method level usually involves changing the code of application. One of the example is the PANACEA framework. It is based on the concept of the *self-healing annotations* used to decorate the objects at the development time. Through these annotations the application developer passes hints to run-time PANACEA healers, which may use them on-demand. The second option is to use AOP (please refer to Section 3.1). It does not require code changes but it involves good knowledge of the system source code and it can be mainly used for gathering the data without notifying the monitored system.

As mentioned above each solution has its *pros* and *cons*. Monitoring the system parameters (CPU usage, disk usage, network bandwidth) can be done without extending the application. On the application level the AOP can be considered together with exposing additional monitoring information – e.g. with the JMX technology beans can be used. For handling the *healing* feedbacks from

the monitoring system, event based communication can be used (in case of JMX, it can be *notifications*).

4.4 Prototype - AgeMon

A prototype of a new *self-healing* system, called AgeMon[3], using the above concept is being developed. It is written in Java and as mentioned it uses JGroups as a communication layer. We have currently implemented the *regular monitoring role* and *GUI role*. The monitoring role is using JMX as a connector to the monitored system. Each *capability* that can be monitored is dynamically read using JMX and converted to the abstract internal description and presented to the user in a Swing based GUI. Owing to it, GUI is able to graphically manage the group of the agents, define and run a new monitoring session and manage the flow of the monitoring results (agent-to-agent connections).

Currently the *database* and *rule* role is under development. The database role will be based on 'in-memory' database (like Apache Derby). The rules and actions can be based on Drools[4] notation or stored in the ontology [16].

4.5 Summary

In this paper we presented the existing technologies that can be used for enabling self-healing which goes beyond pure monitoring. Two of them are based on the Aspect Oriented Programming – by using this technology we can enable self-healing for the systems on the software level. Moreover, it is possible to enable self-healing for legacy applications as well. An example of hardware level self-healing may be Solaris 10 OS.

The second part of the paper focused on the requirements for and our approach to building a self-healing monitoring system. A brief description of the prototype of such a system is presented. The prototype is a first stage for implementing the self-healing monitoring system. We have introduced a concept of roles together with the description of each role. The stack-based agent was presented.

At the moment in the AgeMon system, the regular monitoring role and the GUI role is implemented, so there is also some work to be done in further research – we need to implement other roles like the database role or rule engine to make the system complete.

Acknowledgements. This research is partially supported by the POIG project "PL-Grid" and the AGH grant 11.11.120.865.

References

1. Predictive Self-Healing in the Solaris 10 Operating System - A Technical Introduction (September 2004), http://www.sun.com/bigadmin/content/selfheal/selfheal_overview.pdf

[3] AgeMon stands for *Age*nt-based *Mon*itoring System.
[4] jboss.org/drools.

2. Alonso, J., Torres, J., Silva, L.M., Griffith, R., Kaiser, G.: Towards Self-adaptable monitoring framework for self-healing, CoreGRID TR-0150, July 3 (2008), http://www.coregrid.net/mambo/images/stories/-TechnicalReports/tr-0150.pdf
3. Griffith, R., Kaiser, G.: Adding self-healing capabilities to the common language runtime. Technical report, Columbia University (2005)
4. Using Nagios to monitor faults in a self-healing environment, by Mikko A.T. Pervilä (2007), http://www.cs.helsinki.fi/u/niklande/opetus/SemK07/-paper/pervila.pdf
5. Amin, M.: Toward self-healing energy infrastructure systems. Computer Applications in Power 14(1), 20–28 (2001)
6. Sidiroglou, S., Laadan, O., Keromytis, A.D., Nieh, J.: Using Rescue Points to Navigate Software Recovery (Short Paper). In: Proceedings of the IEEE Symposium on Security and Privacy (May 2007)
7. The Intelligent Software Agents Lab – Home Page, http://www.cs.cmu.edu/~softagents/intro.htm
8. Baresi, L., Guinea, S., Pasquale, L.: Self-healing BPEL Processes with Dynamo and the JBoss Rule Engine. In: Int. Workshop on Engineering of Software Services for Pervasive Environments: in Conjunction with the 6th ESEC/FSE Joint Meeting, Dubrovnik, Croatia, pp. 11–20 (2007)
9. Glassbox – How It Works, http://www.glassbox.com/glassbox/HowItWorks.html
10. AspectJ – Home page, http://www.eclipse.org/aspectj/
11. Guttman, E., Perkins, C., Veizades, J., Day, M.: RFC 2608 Service Location Protocol, Version 2 (June 1999), http://tools.ietf.org/html/rfc2608
12. Goland, Y.Y., Cai, T., Leach, P., Gu, Y., Albright, S.: Simple Service Discovery Protocol/1.0. (October 28, 1999), http://coherence.beebits.net/chrome/site/draft-cai-ssdp-v1-03.txt
13. Jini Discovery and Join Specification v3. September 4 (2006), http://www.jini.org/wiki/Jini_Discovery_and_Join_Specification
14. PANACEA - Towards a Self-healing Development Framework. In: 10th IFIP/IEEE International Symposium on Integrated Network Management, IM 2007, May 21, pp. 169–178 (2007), ISBN: 1-4244-0798-2
15. HP Open View Self-Healing Services: Overview and Technical Introduction, HP Labs (2006), http://managementsoftware.hp.com/services/selfhealing_whitepaper.pdf
16. Funika, W., Godowski, P., Pęgiel, P.: A Semantic-Oriented Platform for Performance Monitoring of Distributed Java Applications. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part III. LNCS, vol. 5103, pp. 233–242. Springer, Heidelberg (2008)
17. Wuttke, J.: An approach to detecting failures automatically. In: Fourth International Workshop on Software Quality Assurance: in Conjunction With the 6th ESEC/FSE Joint Meeting, Dubrovnik, Croatia, pp. 17–24 (2007)

Parallel Performance Evaluation of MIC(0) Preconditioning Algorithm for Voxel μFE Simulation

Ivan Lirkov[1], Yavor Vutov[1], Marcin Paprzycki[2], and Maria Ganzha[2]

[1] Institute for Parallel Processing, Bulgarian Academy of Sciences,
Acad. G. Bonchev, Bl. 25A, 1113 Sofia, Bulgaria
ivan@parallel.bas.bg, yavor@parallel.bas.bg
http://parallel.bas.bg/~ivan/, http://parallel.bas.bg/~yavor/
[2] Systems Research Institute, Polish Academy of Sciences
ul. Newelska 6, 01-447 Warsaw, Poland
paprzyck@ibspan.waw.pl, maria.ganzha@ibspan.waw.pl
http://www.ibspan.waw.pl/~paprzyck/, http://www.ganzha.euh-e.edu.pl

Abstract. Numerical homogenization is used for up-scaling of a linear elasticity tensor of strongly heterogeneous micro-structures. Utilized approach assumes presence of a periodic micro-structure and thus periodic boundary conditions. Rotated trilinear Rannacher-Turek finite elements are used for the discretization, while a parallel PCG method is used to solve arising large-scale systems with sparse, symmetric, positive semidefinite matrices. Applied preconditioner is based on modified incomplete Cholesky factorization MIC(0).

The test problem represents a trabecular bone tissue, and takes into account only the elastic response of the solid phase. The voxel micro-structure of the bone is extracted from a high resolution computer tomography image. Numerical tests performed on parallel computers demonstrate the efficiency of the developed algorithm.

Keywords: micro finite element simulation, modeling of human bone tissue, parallel algorithms, PCG method, preconditioner, MIC(0) factorization, parallel performance.

1 Introduction

Many, seemingly different materials, such as human bone tissue, geocomposites, filtering media in industrial applications have very complex hierarchical organization spanning multiple length scales and involve complex multi-physical processes at some of these scales. However, their overall mechanical response and ability to conduct fluids can be described using multilevel techniques that are built upon basic conservation principles at the micro or nano levels.

In our work, we consider modeling of human bone tissue which is based on the recently developed morphology of bones. In general, model used here has a multilevel structure according to the specific material dimensions (and as such

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 135–144, 2010.

generalizes to other problems mentioned above). At a length scale of about several hundred nanometers, oriented, highly organized collagen molecules, the minority of the hydroxy-apatite crystals (present in bone tissues) and water, build up (mineralized) fibrils. At the same length scale, but in the extra-fibrillar space, the majority of (largely disordered) hydroxy-apatite crystals build up a mineral foam (polycrystalline), with water filling the inter-crystalline (nano)space. At a length scale of several micrometers, the fibrils and the extra-fibrillar space builds up solid bone matrix or ultrastructure. Finally, at a length scale of several millimeters, macroscopic bone material (cortical or trabecular bone) comprises solid bone matrix and the micro-porous space. This four-level model has been validated by statistically and physically independent experiments, see e.g. [11,13,15]. Having in mind that the aforementioned dosages (concentrations, volume fractions) are dependent on complex biochemical control cycles (defining the metabolism of the organism), the purely mechanical theory can be linked to biology, biochemistry, and, on the applied side, to clinical practice.

Many problems in bone modeling result in need to solve large- and very large-scale linear systems. This, in turn, requires application of parallel computers. Furthermore, even though recent advances in direct solvers for large-scale sparse linear systems has to be acknowledged (see, [7,17], for an interesting comparison), the method of choice for the problem at hand has to be iterative.

In this context, this study concerns development and tuning of solution methods, algorithms, and software tools for micro finite element (μFE) simulation of human bones (e.g. [1,2]). Furthermore, the isotropic linear elasticity model considered here is a brick in the development of a generalized toolkit for μFE simulation of the bone micro-structure.

A boundary value problem can be discretized in various ways. Among the most popular are: the finite volume method, the Galerkin finite element method (FEM), and the mixed FEM. Many engineering problems need very accurate velocity (flux) determinations in the presence of heterogeneities and large jumps in the coefficient. This can be achieved through the mixed FEM. However, this technique usually leads to an algebraic saddle point problem that is more difficult and more expensive to solve. An important discovery of Arnold and Brezzi [4] is that the Schur system for the Lagrange multipliers can be obtained also as a discretization by a Galerkin method using nonconforming elements. The application of rotated trilinear hexahedral FEs is studied in this paper.

The resulting linear system is large, with a sparse, symmetric and positive definite matrix. This implies use of preconditioned conjugate gradient (PCG) solvers [5], while choice of preconditioning is crucial for the PCG performance. It is also known that the PCG method converges for semidefinite matrices in the orthogonal to the kernel subspace.

The elasticity stiffness matrix has a coupled block structure corresponding to separable displacement ordering of the unknowns. Until now, the displacement decomposition (see, [6,10]) remains one of the most robust approaches for preconditioning of such matrices. Here, one of the most popular and the most successful class of preconditioners is the class of incomplete LU (ILU) factorizations (see,

e.g. [5,12]). However, one potential problem with the ILU preconditioners is that they exhibit a limited degree of parallelism. To alleviate this problem we have developed a preconditioning algorithm based on a parallel MIC(0) (Modified Incomplete Cholesky) elasticity solver [20]. Suitable modification of the MIC(0) algorithm allows efficient parallelization of the preconditioning.

2 Homogenization Technique

Let Ω be a hexagonal domain representing our reference volume element (RVE) and $\mathbf{u} = (u_1, u_2, u_3)$ be the displacements vector in Ω. Here, components of the small strain tensor are:

$$\varepsilon_{ij}\left(\mathbf{u}\left(\mathbf{x}\right)\right) = \frac{1}{2}\left(\frac{\partial u_i(\mathbf{x})}{\partial x_j} + \frac{\partial u_j(\mathbf{x})}{\partial x_i}\right) \tag{1}$$

We assume that Hooke's law holds:

$$\begin{bmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{23}\\ \sigma_{13}\\ \sigma_{12}\end{bmatrix} = \begin{bmatrix} c_{1111} & c_{1122} & c_{1133} & c_{1123} & c_{1113} & c_{1112}\\ c_{2211} & c_{2222} & c_{2233} & c_{2223} & c_{2213} & c_{2212}\\ c_{3311} & c_{3322} & c_{3333} & c_{3323} & c_{3313} & c_{3312}\\ c_{2311} & c_{2322} & c_{2333} & c_{2323} & c_{2313} & c_{2312}\\ c_{1311} & c_{1322} & c_{1333} & c_{1323} & c_{1313} & c_{1312}\\ c_{1211} & c_{1222} & c_{1233} & c_{1223} & c_{1213} & c_{1212}\end{bmatrix} \begin{bmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ \varepsilon_{33}\\ 2\varepsilon_{23}\\ 2\varepsilon_{13}\\ 2\varepsilon_{12}\end{bmatrix}. \tag{2}$$

Here, tensor c is called the stiffness tensor, while σ is the stress tensor.

The symmetric 6×6 matrix C is called the stiffness matrix. For an isotropic material C has only two independent degrees of freedom. For materials containing three orthogonal planes of symmetry, matrix C has nine independent degrees of freedom: three Young's moduli E_1, E_2, E_3, three Poisson's ratios $\nu_{12}, \nu_{23}, \nu_{31}$ and three shear moduli $\mu_{12}, \mu_{23}, \mu_{31}$.

$$C = \delta \begin{bmatrix} \frac{1-\nu_{23}\nu_{32}}{E_1} & \frac{\nu_{12}+\nu_{31}\nu_{23}}{E_1} & \frac{\nu_{31}+\nu_{21}\nu_{32}}{E_1} & & & \\ \frac{\nu_{12}+\nu_{13}\nu_{32}}{E_2} & \frac{1-\nu_{31}\nu_{13}}{E_2} & \frac{\nu_{32}+\nu_{31}\nu_{12}}{E_2} & & & \\ \frac{\nu_{13}+\nu_{12}\nu_{23}}{E_3} & \frac{\nu_{23}+\nu_{13}\nu_{21}}{E_3} & \frac{1-\nu_{12}\nu_{21}}{E_3} & & & \\ & & & \frac{\mu_{23}}{\delta} & & \\ & & & & \frac{\mu_{31}}{\delta} & \\ & & & & & \frac{\mu_{12}}{\delta} \end{bmatrix}, \tag{3}$$

where

$$\delta = 1 - \nu_{12}\nu_{21} - \nu_{13}\nu_{31} - \nu_{23}\nu_{32} - 2\nu_{12}\nu_{23}\nu_{31},$$

$$\frac{\nu_{12}}{E_1} = \frac{\nu_{21}}{E_2}, \qquad \frac{\nu_{23}}{E_2} = \frac{\nu_{32}}{E_3}, \qquad \frac{\nu_{31}}{E_3} = \frac{\nu_{13}}{E_1}.$$

Our goal was to obtain homogenized material properties of the trabecular bone tissue. In other words, to find the stiffness tensor of a homogeneous material, with the same macro-level properties as our RVE. In the proposed approach, we follow the numerical up-scaling method from [14] (see also [9]). The homogenization scheme requires finding functions $\boldsymbol{\xi}^{kl} = (\xi_1^{kl}, \xi_2^{kl}, \xi_3^{kl})$, $k, l = 1, 2, 3$, satisfying the following problem in a week formulation:

$$\int_{\Omega} \left(c_{ijpq}(x) \frac{\partial \xi_p^{kl}}{\partial x_q} \right) \frac{\partial \phi_i}{\partial x_j} d\Omega = \int_{\Omega} c_{ijkl}(x) \frac{\partial \phi_i}{\partial x_j} d\Omega, \tag{4}$$

for an arbitrary Ω-periodic variational function $\boldsymbol{\phi} \in H^1(\Omega)$. After computing the characteristic displacements $\boldsymbol{\xi}^{kl}$, from (4) we can compute the homogenized elasticity tensor c^H using the following formula:

$$c_{ijkl}^H = \frac{1}{|\Omega|} \int_{\Omega} \left(c_{ijkl}(x) - c_{ijpq}(x) \frac{\partial \xi_p^{kl}}{\partial x_q} \right) d\Omega. \tag{5}$$

Due to symmetry of the stiffness tensor c, we have the relation $\boldsymbol{\xi}^{kl} = \boldsymbol{\xi}^{lk}$ and it is enough to solve only six problems (4) to obtain the homogenized stiffness tensor.

Rotated trilinear (Rannacher-Turek) finite elements [18] are used for the numerical solution of (4). This choice is motivated by the additional stability of the nonconforming finite element discretization in the case of strongly heterogeneous materials [4]. Construction of a robust non-conforming finite element method is generally based on application of mixed formulation leading to a saddle-point system. By the choice of non continuous finite elements for the dual (pressure) variable, it can be eliminated at the (macro)element level. As a result we obtain a symmetric positive (semi-)definite finite element system in primal (displacements) variables. We utilize this approach, which is referred as the *reduced and selective integration* (RSI).

3 Parallel Displacement Decomposition MIC(0) Preconditioning

A preconditioning algorithm was developed using a parallel MIC(0) elasticity solver [20], based on a parallel MIC(0) solver for the scalar elliptic problem [3]. The preconditioner uses the isotropic variant of the displacement decomposition (DD) [6,10]. We write the DD auxiliary matrix in the form

$$M_{DD} = \begin{bmatrix} A & & \\ & A & \\ & & A \end{bmatrix} \tag{6}$$

where A is the stiffness matrix corresponding to the bilinear form

$$a(u^h, v^h) = \int_{\Omega} E(\mathbf{x}) \left(\sum_{i=1}^{3} \frac{\partial u}{\partial x_i} \frac{\partial v}{\partial x_i} \right) d\mathbf{x}, \tag{7}$$

and u and v are Ω-periodic functions. Such approach is motivated by the second Korn's inequality, which holds for the RSI finite element discretization under consideration. This means that the estimate for the relative condition number of the preconditioned system

$$\kappa\left(M_{DD}^{-1}K\right) = \mathcal{O}\left((1-2\nu)^{-1}\right)$$

holds uniformly with respect to the mesh size parameter, while K is the stiffness matrix. Our preconditioner is obtained by the MIC(0) factorization of blocks in (6).

Remark 1. To satisfy conditions for the stable MIC(0) factorization in the case of a semi-definite matrix, we are using the perturbed version of the MIC(0) algorithm, where the incomplete factorization is applied to the matrix $\tilde{A} = A + \tilde{D}$. The diagonal perturbation $\tilde{D} = \tilde{D}(\xi) = diag(\tilde{d}_1, \ldots \tilde{d}_N)$ is defined as follows:

$$\tilde{d}_i = \begin{cases} \xi a_{ii} & \text{if } a_{ii} \geq 2w_i \\ \xi^{1/2} a_{ii} & \text{if } a_{ii} < 2w_i \end{cases}$$

where $0 < \xi < 1$ is a properly chosen parameter, and $w_i = \sum_{j>i} -a_{ij}$.

The idea of the proposed parallel algorithm is to apply the MIC(0) factorization on an auxiliary matrix B, which approximates A. This matrix B has a special block structure, which facilitates implementation of a scalable parallel solver. Following the standard FEM assembling procedure we write A in the form $A = \sum_{e \in \omega_h} R_e^T A_e R_e$, where A_e is the element stiffness matrix, while R_e stands for the restriction mapping of the global vector of unknowns to the local one corresponding to the current element e. Let us consider the following approximation B_e of A_e:

$$A_e = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \end{bmatrix} \qquad B_e = \begin{bmatrix} b_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & b_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & b_{33} & 0 & 0 & 0 \\ a_{41} & a_{42} & 0 & b_{44} & 0 & 0 \\ a_{51} & a_{52} & 0 & 0 & b_{55} & 0 \\ a_{61} & a_{62} & 0 & 0 & 0 & b_{66} \end{bmatrix}$$

Local numbering applied here follows the pairs of the opposite nodes of the reference element. Here, diagonal entries of B_e are modified to hold the row-sum criteria (for more details see [3]). Assembling the locally defined matrices B_e we obtain the global matrix $B = \sum_{e \in \omega_h} R_e^T B_e R_e$. The condition number estimate $\kappa(B^{-1}A) \leq 3$ holds uniformly with respect to the mesh parameter and to possible coefficient jumps (for the related analysis see discussion presented in [3]). After this modification we obtain matrix B with its diagonal blocks (corresponding to horizontal cross sections) being diagonal matrices. The solution of linear systems with the preconditioner can be performed in parallel. It is important to note that the use of periodic boundary conditions does not change diagonal blocks of the

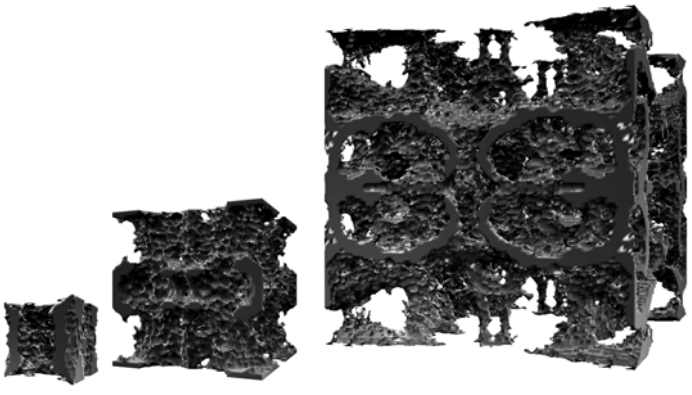


Fig. 1. Structure of the solid phase: (a) $32 \times 32 \times 32$ voxels, (b) $64 \times 64 \times 64$ voxels, (c) $128 \times 128 \times 128$ voxels

stiffness matrix A, and thus the same parallel algorithm can be applied also here. The obtained preconditioner has the form.

$$M_{DDMIC(0)} = \begin{bmatrix} M_{MIC(0)}(B) & & \\ & M_{MIC(0)}(B) & \\ & & M_{MIC(0)}(B) \end{bmatrix}.$$

4 Experimental Results

Our test specimen is part of a trabecular bone tissue obtained from a high resolution computer tomography [8]. The trabecular bone tissue has a strongly heterogeneous micro-structure composed of solid and fluid phases and thus matches very well with the proposed approach. To obtain a periodic RVE the bone tissue specimen is mirrored three times, see Fig. 1. The voxel size is 37 μm.

In this paper we focus on the parallel performance of the proposed numerical up-scaling technique. Experiments to study the homogenized properties of the RVE with dimensions $n \times n \times n$ voxels were performed, where $n = 32, 64, 128$. The Young moduli $E^s = 14.7$ GPa for the solid phase, and $E^f = 1.323$ GPa for the fluid phase were used. The same Poisson ratio $\nu^s = \nu^f = 0.325$ was used for both phases. The iteration stopping criterion was $||\mathbf{r}^j||_{M^{-1}}/||\mathbf{r}^0||_{M^{-1}} < 10^{-3}$, where $\mathbf{r}^j$ stands for the residual at the j-th iteration step of the preconditioned conjugate gradient method.

To solve the above described problems, a portable parallel FEM code was designed and implemented in C++, while the parallelization has been facilitated using the MPI library [19,21]. The parallel code has been tested on cluster computer system located in the Oklahoma Supercomputing Center (OSCER) and the IBM Blue Gene/P machine at the Bulgarian Supercomputing Center. In our experiments, times have been collected using the MPI provided timer and

Table 1. Experimental results on Sooner

n	Problem size	p	Time	Speed-up	Efficiency
32	2 359 296	1	631.05		
		2	355.73	1.77	0.887
		4	243.61	2.59	0.648
		8	217.92	2.90	0.362
		16	133.24	4.74	0.296
		32	87.86	7.18	0.224
64	18 874 368	1	6317.35		
		2	3362.10	1.88	0.939
		4	2270.16	2.78	0.696
		8	1921.95	3.29	0.411
		16	1154.40	5.47	0.342
		32	774.10	8.16	0.255
		64	588.81	10.73	0.168
128	150 994 944	1	82468.90		
		2	44902.69	1.84	0.918
		4	28065.57	2.94	0.735
		8	23621.77	3.49	0.436
		16	12146.39	6.79	0.424
		32	7212.52	11.43	0.357
		64	4881.75	16.89	0.264
		128	3761.78	21.92	0.171

we report the best results from multiple runs. We report the elapsed time T_p in seconds on p processors, the parallel speed-up $S_p = T_1/T_p$, and the parallel efficiency $E_p = S_p/p$. The obtained up-scaled properties can be found in [16].

Table 1 summarizes results collected on Sooner. It is a Dell Pentium4 Xeon E5405 ("Harpertown") quad core Linux cluster located in the Oklahoma Supercomputing Center (see `http://www.oscer.ou.edu/resources.php`). It has 486 Dell PowerEdge 1950 III nodes and two quad core processors per node. Each processor runs at 2 GHz. Processors within each node share 16 GB of memory, while nodes are interconnected with a high-speed InfiniBand network. We have used Intel C++ compiler and compiled the code with the following options: "-O3 -march=core2 -mtune=core2".

As expected, the parallel efficiency improves with the size of the discrete problems. The efficiency on 16 processors increases from 30% for the smallest problems to 42% for the largest problems in this set of experiments. Also, the execution times decrease with increasing number of processors which shows that the communications in our parallel algorithm are mainly local.

Table 2 shows execution times on IBM Blue Gene/P machine at the Bulgarian Supercomputing Center (see `http://www.scc.acad.bg/`). It consists of 2048 compute nodes with quad core PowerPC 450 processors (running at 850 MHz). Each node has 2 GB of RAM. For the point-to-point communications a 3.4 Gb 3D mesh network is used. Reduction operations are performed on a 6.8 Gb tree

Table 2. Experimental results on IBM Blue Gene/P

n	Problem size	p	Time	Speed-up	Efficiency
32	2 359 296	1	3442.29		
		2	1782.88	1.93	0.965
		4	954.90	3.61	0.901
		8	532.29	6.47	0.808
		16	322.62	10.67	0.667
		32	205.40	16.76	0.524
64	18 874 368	1	34166.01		
		2	17358.17	1.97	0.984
		4	8975.65	3.81	0.952
		8	4763.87	7.17	0.896
		16	2633.40	12.97	0.811
		32	1589.88	21.49	0.672
		64	1003.19	34.06	0.532
128	150 994 944	8	55413.21		
		16	29132.86		0.951
		32	15967.65		0.868
		64	9773.76		0.709
		128	6131.38		0.565

network. We have used IBM XL C++ compiler and compiled the code with the following options: "-O5 -qstrict -qarch=450d -qtune=450".

The memory available on a single node of Blue Gene/P is not sufficient to run experiments for the largest problem and we report execution times starting from eight processors located within different nodes. Therefore, for the largest problem, we report parallel efficiency related to results collected on eight processors. Execution times on Blue Gene/P are substantially larger than that on Sooner,

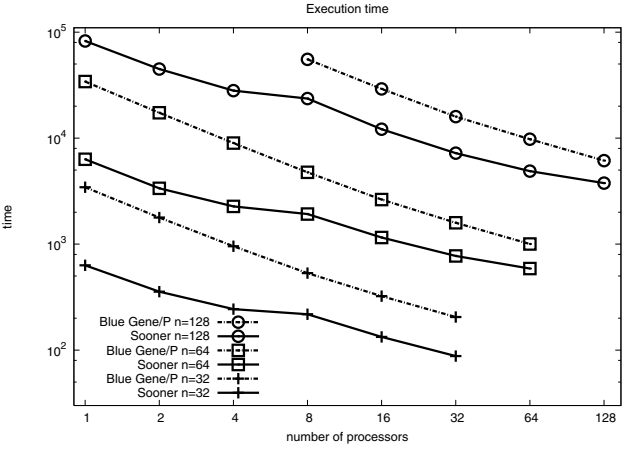


Fig. 2. Execution times for the test problems

but parallel efficiency obtained on the supercomputer is better. For instance, the execution on 64 processors on Sooner is only twice faster than on the Blue Gene/P, in comparison with five times faster performance on single processor.

To summarize, in Fig. 2 computing times on both parallel systems are shown. The somehow slower performance on Sooner using 8 processors is clearly visible. It can be stipulated that this effect is a result of limitations of memory subsystems and their hierarchical organization. One of them might be the limited bandwidth of the main memory bus. This causes processors to "starve" for data, thus, decreasing the overall performance. Note that L2 cache memory is shared among each pair of cores within the processors of Sooner. This boosts performance of programs utilizing only a single core within such pair. Conversely, this leads to somewhat decreased performance when all cores are used.

5 Conclusions and Future Work

We have studied the parallel performance of the recently developed numerical homogenization technique utilizing parallel MIC(0) factorization [16]. The performance was evaluated on two different parallel architectures. Satisfying parallel efficiency is obtained on the IBM Blue Gene/P. The efficiency on Sooner quickly deteriorates with the increase of the number of the processors. Despite of the worse efficiency, the faster CPUs on Sooner lead to shorter runtime, on the same number of processors. The network latency is crucial for the parallel performance of the algorithm. To hide some of the network latency, the computations were overlapped with the communications wherever possible. We plan to investigate the possibility to hide further the latency, solving simultaneously more than one of the six independent problems (4).

Acknowledgments

Computer time grants from the Oklahoma Supercomputing Center (OSCER) and the Bulgarian Supercomputing Center (BGSC) are kindly acknowledged. This research was partially supported by grant DO02-147/2008 from the Bulgarian NSF. Work presented here is a part of the Poland-Bulgaria collaborative grant "Parallel and distributed computing practices".

References

1. Arbenz, P., Flaig, C.: On smoothing surfaces in voxel based finite element analysis of trabecular bone. In: Lirkov, I., Margenov, S., Waśniewski, J. (eds.) LSSC 2007. LNCS, vol. 4818, pp. 69–77. Springer, Heidelberg (2008)
2. Arbenz, P., Van Lenthe, G.H., Mennel, U., Muller, R., Sala, M.: A scalable multilevel preconditioner for matrix-free μ-finite element analysis of human bone structures. Internat. J. Numer. Methods Engrg. 73(7), 927–947 (2008)
3. Arbenz, P., Margenov, S., Vutov, Y.: Parallel MIC(0) preconditioning of 3D elliptic problems discretized by Rannacher-Turek finite elements. Computers and Mathematics with Applications 55(10), 2197–2211 (2008)

4. Arnold, D.N., Brezzi, F.: Mixed and nonconforming finite element methods: Implementation, postprocessing and error estimates. RAIRO, Model. Math. Anal. Numer. 19, 7–32 (1985)
5. Axelsson, O.: Iterative solution methods. Cambridge Univ. Press, Cambridge (1994)
6. Axelsson, O., Gustafsson, I.: Iterative methods for the solution of the Navier equations of elasticity. Comp. Meth. Appl. Mech. Eng. 15, 241–258 (1978)
7. Bängtsson, E., Lund, B.: A comparison between two solution techniques to solve the equations of glacially induced deformation of an elastic earth. Internat. J. Numer. Methods Engrg. 75, 479–502 (2008)
8. Beller, G., Burkhart, M., Felsenberg, D., Gowin, W., Hege, H.-C., Koller, B., Prohaska, S., Saparin, P.I., Thomsen, J.S.: Vertebral body data set esa29-99-l3, http://bone3d.zib.de/data/2005/ESA29-99-L3/
9. Bensoussan, A., Lions, J.L., Papanicolaou, G.: Asymptotic analysis for periodic structures. Elsevier, Amsterdam (1978)
10. Blaheta, R.: Displacement decomposition–incomplete factorization preconditioning techniques for linear elasticity problems. Num. Lin. Alg. Appl. 1(2), 107–128 (1994)
11. Fritsch, A., Dormieux, L., Hellmich, C., Sanahuja, J.: Micromechanics of crystal interfaces in polycrystalline solid phases of porous media: fundamentals and application to strength of hydroxyapatite biomaterials. J. Materials Science 42(21), 8824–8837 (2007)
12. Golub, G.H., Van Loan, C.F.: Matrix computations, 2nd edn. Johns Hopkins Univ. Press, Baltimore (1989)
13. Hellmich, C., Kober, C.: Micromechanics-supported conversion of computer tomography (CT) images into anisotropic and inhomogeneous FE models of organs: the case of a human mandible. Proceedings in Applied Mathematics and Mechanics 6, 71–74 (2006)
14. Hoppe, R.H.W., Petrova, S.I.: Optimal shape design in biomimetics based on homogenization and adaptivity. Math. Comput. Simul. 65(3), 257–272 (2004)
15. Kober, C., Erdmann, B., Hellmich, C., Sader, R., Zeilhofer, H.-F.: Consideration of anisotropic elasticity minimizes volumetric rather than shear deformation in human mandible. Comput. Meth. Biomech. Biomedic. Engin. 9(2), 91–101 (2006)
16. Margenov, S., Vutov, Y.: Parallel MIC(0) preconditioning for numerical upscaling of anisotropic linear elastic materials. In: Lirkov, I., Margenov, S., Waśniewski, J. (eds.) LSSC 2009. LNCS, vol. 5910, pp. 805–812. Springer, Heidelberg (2010)
17. Neytcheva, M., Bängtsson, E.: Preconditioning of nonsymmetric saddle point systems as arising in modelling of visco-elastic problems. ETNA 29, 193–211 (2008)
18. Rannacher, R., Turek, S.: Simple nonconforming quadrilateral Stokes element. Numer. Methods for Partial Differential Equations 8(2), 97–112 (1992)
19. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: the complete reference. Scientific and engineering computation series. The MIT Press, Cambridge (1997) (second printing)
20. Vutov, Y.: Parallel DD-MIC(0) preconditioning of nonconforming rotated trilinear FEM elasticity systems. In: Lirkov, I., Margenov, S., Waśniewski, J. (eds.) LSSC 2007. LNCS, vol. 4818, pp. 745–752. Springer, Heidelberg (2008)
21. Walker, D., Dongarra, J.: MPI: a standard Message Passing Interface. Supercomputer 63, 56–68 (1996)

Parallel HAVEGE

Alin Suciu[1], Tudor Carean[1], Andre Seznec[2], and Kinga Marton[1]

[1] Technical University of Cluj-Napoca, Cluj-Napoca, Romania
{alin.suciu,tudor.carean,kinga.marton}@cs.utcluj.ro
[2] IRISA/INRIA, Rennes, France
andre.seznec@irisa.fr

Abstract. The HAVEGE algorithm [1] [2] generates unpredictable random numbers by gathering entropy from internal processor states that are inheritably volatile and impossible to tamper with in a controlled fashion by any application running on the target system. The method used to gather the entropy implies that its main loop will almost monopolize the CPU; the output depends on the operating system and other running applications, as well as some internal mechanisms that stir the processor states to generate an enormous amount of entropy. The algorithm was designed with the idea of single-core CPUs in mind, and no parallelization; however the recent market explosion of multi-core CPUs and the lack of results in increasing the CPU frequency justifies the need to research a multithreaded parallel version of HAVEGE, capable of running the same algorithm loop on each core independently and transparently combine the results in one single output bitstream. This paper will demonstrate how such a parallelization is possible and benchmark the output speed of its implementation.

Keywords: random number generator, HAVEGE, parallel implementation.

1 Introduction

The HAVEG (HArdware Volatile Entropy Gathering) algorithm family [1] indirectly gathers entropy produced by external sources in the internal processor states using the memory hierarchy and the branch prediction mechanism. The algorithm uses the hardware clock counter to indirectly extract entropy from the otherwise invisible hardware states.

The HAVEGE (HArdware Volatile Entropy Gathering and Expansion) generator [1] [2] combines a HAVEG algorithm of entropy extraction with a pseudo-random number generator (two concurrent walks on a self modifying table). The result is both a high bit rate as well as a high security level.

The security of the algorithm relies upon the fact that one can not completely determine the internal state of the generator because this state is not only composed of memory mapped data but of thousands volatile hardware states that are inaccessible even to the user running the application.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 145–154, 2010.

There is no direct way to read these internal states without using special debugging motherboards and hardware platforms. Any external event such as an indirect attempt to read these states through software would cause a major perturbation in themselves.

The algorithm relies on the hardware clock counter to have an idea of how long it took to execute a particular code sequence. Since instruction execution (number of CPU cycles) depends on many factors, most of them beyond the control and observation area of the user or programmer, it is a good candidate for inserting entropy in a pseudo-random number generator.

The internal workings of the HAVEGE algorithm are detailed in section 2, followed in section 3 by the description of the parallelization solution that we implemented and some experimental results in section 4; in section 5 we present the conclusions.

2 The Internals of the HAVEGE Algorithm

If we consider a short instruction sequence with just a conditional jump and a memory operation and measure how many clock cycles are needed for its execution at different times and possibly different conditions (suppose the content of the variable tested for the jump and/or the memory location accessed changes), different readings will be obtained each time. This is because a memory access does not always need the same number of cycles to complete.

Systems try to avoid reading from a high latency memory such as the RAM as much as possible by using two levels of (faster but much smaller) cache memory. If the requested information is found in the L1 (level 1) cache the read will be completed a lot faster than if it were in L2 (level 2) cache, in RAM or in the worst case scenario in the operating systems swap file (which would require a slow I/O operation).

In addition modern CPUs execute instructions in a pipeline so a read or write could be delayed because of other instructions executing at the same time. A memory access also needs to wait to get control of the bus. Another device besides the CPU could be keeping the bus busy as well by doing a DMA transfer. On the top of this, processors have a lot of internal buffers whose states also influence the execution time in unpredictable ways (at least as seen from the outside).

A simple memory access introduces a lot of uncertainty regarding execution time but conditional jumps introduce even more. In order to implement pipelining, modern CPUs use branch prediction to deal with the uncertainty of the outcome of the conditional test. A decision needs to be made about which side of the jump is to be put in the pipeline even before the test is made.

Execution starts for instructions that may or may not need to run. Based on several factors (that CPU manufacturers dont explain in too much detail) a prediction is made and execution continues with one of the two branches.

If the prediction was wrong everything executed will be undone, the pipeline is emptied, and a jump is made at the right location. If the prediction was right then the CPU has saved a lot of time.

The number of cycles needed for this is unknown and depends on the state of the branch prediction table, the content of the pipeline and the result of the conditional test. One could say that for a test done in a loop the outcome would be the same but this is not always the case.

If the programmer intentionally aggravates things by using a random number (possibly even obtained from past loops) for the test, not only the result is always unknown but it would also confuse the branch predictor that tries to rely on data locality and past results in the same loop.

Modern processors use more and more complex pipelining techniques, meant to improve performance in favorable cases such as consecutive instructions that can be executed independently and out of order, working with different registers or memory locations, without having to wait for each others result. Many other optimizations can be done like register renaming for example. In general the longer the code sequence is the more uncertainty is introduced.

Moreover, interrupts occur at times out of the program's control and these alter the state of the cache memories, content of the pipelines and execution flow in general. Considering a multitasking operating system, other programs, drivers and kernel code need to be executed as well and will interrupt the HAVEGE algorithm for unknown periods of time, which may have a big influence on the internal state of the processor, adding more volatility and therefore entropy to be gathered.

The instruction set of a PC does not offer any method to directly probe all the hardware states. In an effort to break the security of the algorithm perhaps a method could be devised to deduce some of them, but executing additional instructions alters the state of the hardware making a guess useless. The only thing that comes even close to monitoring the volatile hardware states are the debugging platforms (special motherboards) used by processor manufacturers.

To make sure that the generated numbers are as random as possible the algorithm works on a table that is twice the size of the level 1 cache doing two self modifying concurrent walks. The fact that the indexes themselves are random numbers and that the size of the array is twice the size of the level 1 cache gives a 50% probability of a cache miss.

The compiled size of the algorithms loop is also optimized to be as close as possible to the size of the instruction cache, so that any interrupt issued by the operating system would start causing cache misses. Nested if statements are used to exercise the branch prediction mechanism by testing various bits of numbers in the walk table (these numbers are also random).

Every now and then the algorithm reads the hardware clock counter and writes a new value in the output table by combining the entropy just gathered from the clock with values already present in the internal walk table. The iteration continues until the output table is full with random numbers.

3 Parallelization of HAVEGE

To make the algorithm more secure against somebody trying to guess the next sequence one could skip certain results (the HAVEGE authors called this feature *step hiding*) by not returning from the function once the output buffer is full, but instead uses this buffer (and overwrites it) to generate a new set of random numbers.

This could be done several times (several steps), sacrificing speed for security. While the algorithm with no such step hiding is very fast, generating around 350 MB/s of random data, the version recommended for cryptographic purposes that skips 32 result sets, generates no more than 11 MB/s. The latter performance measurement justifies the effort to create a parallel version of this algorithm in order to crease the output rate (speed) of the generator.

Since the algorithm is designed to take over and monitor the buffers, caches and branch prediction table of one processor core the only way to both parallelize and to follow the initial idea behind the algorithm is to run a copy of the algorithm independently on each separate core, each using the dedicated caches and internal mechanisms of its core.

The implementation was done in Visual C++ with a MFC interface that allows benchmarking the algorithms output rate with different settings; the user can decide to store the random numbers on the disk or not.

The original algorithm was not designed with parallelization in mind so the first step was to restructure the code by transforming the rigid functions and global variables into a class that can be instantiated into independent random number generators (objects) that can be used with different settings and don't influence each other. The new version can be run in different threads at the same time and yield the same quality of random data, as long as different cores would be used for each thread.

In order to obtain a unified generator that outputs data at a rate equal to the sum of all the instantiated generators, a master thread needs to synchronize them and gather/use the generated data. Such a master thread needs to tell each generator when to start producing new numbers and when to stop and to make sure that a generator waits enough until its output buffer has been consumed before overwriting it again.

In order to avoid busy waiting and save CPU time the new HAVEGE generator class exposes a set of WIN32 events that the master thread can use to synchronize with the workers. The following pseudo-code explains the algorithm:

```
procedure HavegeGenerator ()
begin
  repeat
      wait for signal to start generating
      run original Havege loop
      signal master thread: buffer is full
  until (shutdown = true)
end
```

```
procedure MasterThread (nbOfHavegeThreads)
  begin
    nbOfPieces := fileSize / bufferSize
    for i:=1 to nbOfHavegeThreads
    begin
      listOfGenerators[i] := new HavegeGenerator()
      run the generator in a new thread
      set the processor affinity mask of the new thread
      signal generator to start generating
    end
    for i:=1 to nbOfPieces
    begin
      wait for a generator to finish
      ind:= index of first generator that finished
      consume the buffer of listOfGenerators[ind]
      signal listOfGenerators[ind] to continue
    end
    shutdown threads
end
```

The resulting program benchmarks the parallelized versions of the two functions provided by the original HAVEGE (*ndrand* and *cryptondrand*), also giving the user the possibility to change parameters like *step hiding*. The user has the option of selecting the number of cores to be used, and the possibility to save the random data in a file for later use or examination.

Among other statistics the program shows the number of cores available on the current machine and calculates the CPU optimization settings the program was compiled with, such as the HAVEGE machine code loop size (that should be as close as possible to the CPUs instruction cache) and the size of the data cache the program is targeting (the HAVEGE walk table should be twice the size of the CPUs data cache).

The application is designed to do two things: gather random data in a file for further quality testing (the *Generate* button) as well to benchmark the speed of the algorithm itself with no disk writing (the *Benchmark* button).

4 Experimental Results

The experiments were aimed at comparing the efficiency between running a serial version of the algorithm (actually the parallel version running on a single core) and a parallel version running on a different number of cores, as well as determining the usefulness of a parallel version in a real life scenario where the output needs to be consumed or stored on disk.

Tables 1 and 2 give the results of the tests on a system with 2 Intel Xeon E5405 at 2 GHz CPUs (8 cores total), 4 GB RAM, running Windows 2008 Server (64

Table 1. Generator speed (in MB/s) without disk writing

Threads/Algorithm	ndrand	cryptondrand	Custom 664	Custom 128
1	476	14.77	7.39	3.69
2	915	29.83	14.90	7.43
3	1383	44.74	22.34	11.17
4	1747	59.74	29.86	14.93
5	2243	74.60	37.29	18.63
6	2661	89.67	44.83	22.40
7	3111	104.46	52.21	26.07
8	3203	119.47	59.66	29.82

Table 2. Generator speed (in MB/s) with disk writing

Threads/Algorithm	ndrand	cryptondrand	Custom 664	Custom 128
1	57.42	13.14	6.96	3.59
2	58.72	26.46	13.92	7.20
3	58.95	39.01	21.08	10.77
4	59.02	52.07	28.07	14.44
5	59.56	61.15	34.12	17.96
6	58.94	60.20	41.37	21.59
7	59.33	59.31	47.20	26.08
8	59.61	59.36	55.08	28.84

bit edition). The speed values in the tables below are the average speeds after generating 100 GB of random data.

Figures 1 and 2 show the results graphically; one can notice the linear increase in speed as well as the limits imposed by writing to disk.

The tests were run both with and without storing the data on the disk to measure the sheer power of the algorithm without capping the output rate by the physical limitations of the hard drive, as well as under more usual conditions where the data needs to be stored.

Increasing the number of threads results in greater speed until we reach the upper limit of 8 threads, because the test system had two quad core CPUs. Any value greater than the number of available cores will give the same performance at best or a slightly lower one due to more inter-thread communication.

Also increasing the number of threads beyond the number of physically independent available cores could have a negative impact on the quality of the random data. Each instance of the algorithm is supposed to work with only one core and its cache memories.

The output speed difference between using 7 and 8 threads is very small because running the master thread (the threads that controls the independent generators and consumes the numbers) on the same core with a generator forces them to compete for CPU time, resulting in decreased performance.

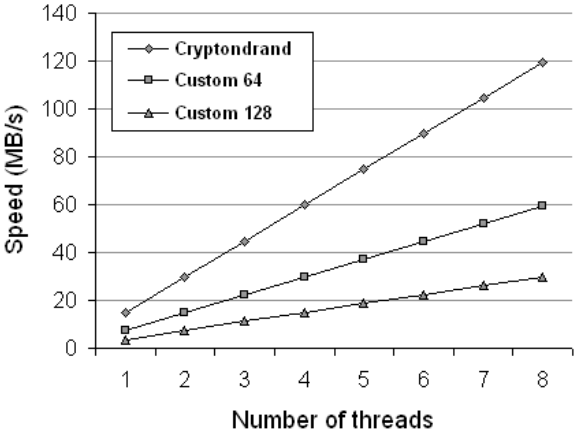


Fig. 1. Generator speed (in MB/s) without disk writing

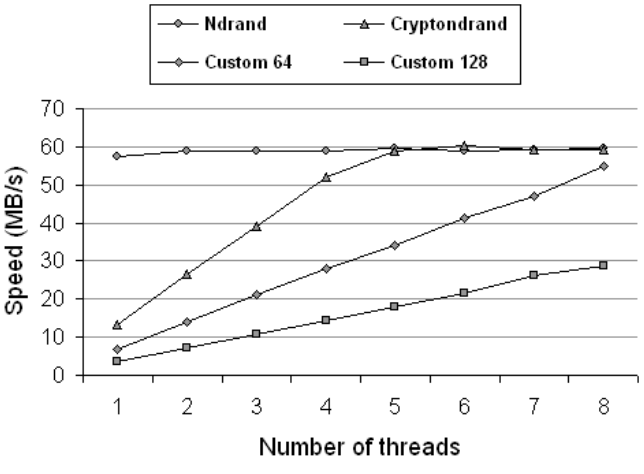


Fig. 2. Generator speed (in MB/s) with disk writing

When testing without disk writing the output speed increases linearly with the number of threads used, as expected, and it decreases linearly with the increase of the *step hiding* parameter. The *ndrand* function uses a *step hiding* value of 1 and the *cryptondrand* function a value of 32. Additionally two custom functions were also tested, one with 64 steps and one with 128 steps.

When writing the data on disk, the output speed is capped by the maximum disk writing speed. As long as the generators can match it, the speed remains almost constant. The differences in values could be attributed to disk fragmentation and background processes doing other I/O operations. In the cases where the generator speed is lower than the disk writing speed, the total output speed is a bit lower than the generator speed.

Table 3. Test results for $\alpha = 0.001$

Test number	ndrand (% tests passed)	cryptondrand (% tests passed)	Quantis (% tests passed)
1	100.00	99.70	99.80
2	99.90	99.80	99.80
3	100.00	99.90	99.90
4	100.00	99.90	99.80
5	99.90	100.00	99.80
7	99.88	99.91	99.92
8	99.60	99.90	99.70
9	100.00	99.90	99.90
11	99.70	99.90	99.90
12	99.90	99.90	99.90
13	99.90	99.90	99.90
14	100.00	99.60	99.90
15	99.83	99.88	99.88
16	99.87	99.92	99.88

Table 4. Test results for $\alpha = 0.01$

Test number	ndrand (% tests passed)	cryptondrand (% tests passed)	Quantis (% tests passed)
1	99.80	99.20	99.00
2	99.60	99.00	98.90
3	99.20	99.10	98.60
4	99.40	98.80	98.90
5	99.00	99.40	98.70
7	99.01	98.99	99.02
8	97.90	97.80	97.90
9	99.10	99.00	99.00
11	98.80	98.90	99.00
12	99.00	98.90	98.90
13	99.10	98.80	98.70
14	99.00	97.80	98.70
15	98.85	99.08	99.05
16	98.98	99.23	98.87

The experiment shows that each generator thread cumulates around 0.1 MB/s overhead, probably due to extra code and time needed to make an I/O call. So when using 8 threads to generate and store the numbers the output speed will be around 1 MB/s smaller than the actual generators speed.

When testing the quality of a random number generator, one or more batteries of statistical tests are usually used [3] [5] [6]; we used the well known NIST Statistical Test Suite, which contains 16 statistical tests [3] [4], of which two are no longer in use (tests 6 and 10).

We compared the results of Parallel HAVEGE's *ndrand* and *cryptondrand* functions with the results given by a quantum based TRNG, called Quantis [7]. The amount of data collected from each source was 1000 MB, grouped in 1000 consecutive sequences of 1 MB each. Parallel HAVEGE was run with 4 parallel generators on a quad core machine (Intel Core 2 Quad 6600 CPU with 2 GB of RAM, running Windows XP).

An important issue here is the choice of α (significance level) which gives the threshold for deciding whether a sequence passed or failed some statistical test. Usual values for this parameter are 0.001 and 0.01, so we tested both these scenarios. For an ideal random number generator, on average, in the first case one should expect 1 sequence in 1000 to fail (99.90 % tests passed) while in the second case one should expect 10 sequences in 1000 to fail (99.00 % tests passed).

The results shown in Table 3 and 4 confirm the fact that Parallel HAVEGE gives a high quality output, which is comparable to a quantum based TRNG. Therefore we may conclude that parallelization did not affect in any way the quality of the original HAVEGE implementation [1].

5 Conclusions

This paper presents a parallel version of the HAVEGE algorithm for generating unpredictable random numbers based on hardware volatile entropy gathering and expansion.

The experimental results show that a parallel version of HAVEGE is useful if a large amount of unpredictable random numbers is needed. The use of any function that is safer (and therefore slower) than the *ndrand* function could also benefit from a speed boost in real life conditions where the numbers need to be used, and usually the consumption rate is higher than the output rate.

The more steps performed by the custom harvesting function, the more volatile values are involved in the generation, the more difficult will be for an adversary to predict the output and therefore the greater the security of the algorithm. However, in a serial implementation this increase in security comes at a price: as we increase the number of steps, the speed (throughput) of the generator will decrease proportionally.

A significant advantage of having a parallel implementation of HAVEGE running on a multicore architecture is the ability to increase the security (number of steps) while maintaining a constant throughput by increasing the number of HAVEGE threads up to the number of available cores.

Acknowledgements

This work was supported by the CNMP funded CryptoRand project (nr. 11-020/2007).

References

1. Seznec, A., Sendrier, N.: HAVEGE: a user-level software heuristic for generating empirically strong random numbers. ACM Transaction on Modeling and Computer Simulations 13(4) (2003)
2. Seznec, A., Sendrier, N.: Hardware Volatile Entropy Gathering and Expansion: generating unpredictable random numbers at user level. In: INRIA Research Report, RR-4592 (2002)
3. Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., Dray, J., Vo, S.: A statistical test suite for random and pseudorandom number generators for cryptographic applications, NIST Special Publication 800-22 (revised August 2008), http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22b.pdf
4. Kim, S., Umeno, K., Hasegawa, A.: Corrections of the NIST Statistical Test Suite for Randomness, Cryptology ePrint Archive, Report 2004/018 (2004)
5. L'Ecuyer, P., Simard, R.: TestU01: A C library for empirical testing of random number generators. ACM Trans. Math. Softw. 33(4), 22 (2007)
6. Knuth, D.E.: The Art of Computer Programming. In: Seminumerical Algorithms, 3rd edn., vol. 2, Addison-Wesley, Reading (1998)
7. IdQuantique, Quantis white paper, http://www.idquantique.com/products/files/quantis-whitepaper.pdf

UNICORE Virtual Organizations System

Krzysztof Benedyczak[1], Marcin Lewandowski[1], Aleksander Nowiński[2], and Piotr Bała[1,2]

[1] Faculty of Mathematics and Computer Science
Nicolaus Copernicus University
Chopina 12/18, 87-100 Toruń, Poland
[2] Interdisciplinary Center for Mathematical and Computational Modelling
Warsaw University
Pawińskiego 5a, 02-106 Warsaw, Poland

Abstract. In this paper we present a novel Virtual Organizations management system called UVOS, developed within the Chemomentum project. It is a complete and production ready solution aiming at simplicity of deployment and management without sacrificing a flexibility and functionality. The system can be also used as an underlying technology for more high level solutions. The system was designed mainly for the UNICORE grid middleware but, as it uses the open SAML protocol and implements numerous SAML profiles, its adoption for other grid middlewares is straightforward. The paper compares the UVOS system to the other existing and popular solutions: VOMS and Shibboleth used with GridShib.

1 Introduction

During the last few years significant effort was put into development of the *Virtual Organizations* concept. Roughly we can divide it into two categories: creation of a VO foundation technologies and high level solutions more concerned about SLA, semi automatic creation of federations etc. Our work clearly belongs to the first field.

The key difficulty in the realization of the Virtual Organizations idea is a storage, access and management of its members and their corresponding privileges. The typical solution which is used in classic IT systems is a *directory service* keeping identities of system users, grouping them and assigning attributes which can be used in an authorization process. The directory services (including the most popular LDAP[1] services) are difficult to be used in a nowadays grid environment because of numerous reasons. Among others it is hard to ensure user's data privacy. Directory service authentication is not compatible with the grid credentials. Therefore dedicated systems were created to support VO management in the grid. The most commonly used are: VOMS [1] and GridShib [2].

[1] LDAP — Lightweight Directory Access Protocol is a *de facto* standard solution for authentication and authorization in nowadays operating systems.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 155–164, 2010.

VOMS is a complete but simple system. It organizes VO members in a pseudo-hierarchical groups and allows for assigning roles. The role can be seen as a single available attribute. VOMS stores grid identities internally (as taken from certificates) and can issue extended GSI (Grid Security Infrastructure) proxy certificates with an additional information about user's roles and group membership, which is called an attribute certificate.

GridShib is an adaptation of the Shibboleth [3] system for the Globus Toolkit. Shibboleth is an advanced federation management system build by educational community. Shibboleth itself does not store user's data but relies on a 3rd party software (eg. LDAP). Shibboleth offers a standardized SAML [4] interface and it is *de facto* a reference SAML implementation. GridShib is clearly focused on an authentication and a support of authorization of web browser users.

One of the principal aims of the Chemomentum project [5] was to create or adopt a VO management solution which will serve the UNICORE grid platform [6]. Naturally the system should be as interoperable as possible and not limited to the UNICORE. After the evaluation of VOMS, Shibboleth and other solutions (like Grid User Management System — GUMS) it turned out that in practice they have a lot of disadvantages that effectively blocks their adoption in many grid deployments. In particular, VOMS it is too proxy-certificates oriented and it is not OGSA [7] enabled. Both those limitations could be eliminated with some effort. However it was difficult to overcome another VOMS inflexibility: poor support for arbitrary attributes (the "generic attributes" concept in VOMS was not fully developed, e.g. the role attribute can be scoped while generic attributes can not), lack of truly hierarchical groups and a missing support for web environments. On the other hand Shibboleth posses a lot of required flexibility but at enormous cost of complexity. In order to create a simple deployment for a small-sized grid it is required to configure Shibboleth service provider, LDAP backend, Signed/Grouper applications for permissions and group management. Still additional modules are needed to provide grid support. Configuration of the whole system is inherently complicated and requires a lot of work and expertise.

As we believe that the problem which must be solved by the low level VO software is not as complicated to sacrifice either flexibility or easy of use we have designed and created a new solution called UVOS – *UNICORE Virtual Organization System*.

1.1 Project Aims

To overcome the problems presented above we have defined a set of concrete goals for the UVOS system:

- *Distributed environment*: the single installation must be *fully* controlled remotely.
- *Openness*: The system consumers must be able to communicate with it using open and well established protocols and data formats. Must not be too UNICORE centric.
- *Easy of Use*: The system needs to be easily installed and managed.

– *Flexibility*: The system should provide tools and features which will make it useful in both OGSA (SOA) and WWW environments. Different deployment styles should be supported.

In the next sections we describe our realization of the above fundamental requirements. Here we only want to stress that our vision was to create a solution which will have a lot to offer out-of-the box and still will be an interesting platform for more high level developments.

2 Architecture

UVOS architecture uses a central UVOS server which acts both as authentication service and attribute authority. The server is used by two kinds of clients: consumers and management clients. Consumers do not modify the UVOS contents but query it. Management clients are used either by VO administrators or by other management software to dynamically modify VO data. What is worth to note here is that consumers use the standard and open SAML 2.0 protocol only.

The server uses relational database to store internally the whole data. Therefore UVOS does not depend on any external service like LDAP. Relational database engine can be easily changed as database access is performed through the iBATIS library [8]. Typically embedded HSQL [9] database is used so it is not needed to configure a standalone DBMS.

Network server is based on the embedded Jetty server. The UVOS server (as the whole system) is written purely in Java so it is highly portable. All operations of the UVOS server are available via the web services interface. The server supports multiple configurable authentication mechanisms (HTTP DIGEST Authorization and TLS authentication with client's certificate) so high flexibility is achieved. Except of the authentication with X.509 certificate, it is also possible to provide simple email and password credentials, what is especially useful in WWW environments. The UVOS server also supports Explicit Trust Delegation (ETD) [10] which allows for better integration with the UNICORE grid by using delegated credentials. Using the ETD the UNICORE service can ask for user's attributes on this user's behalf and UVOS server will authorize the caller as in case of a self query.

The web service interface defines multiple port types, but conceptually we can divide them into the two parts: a management part which uses a custom protocol and a query part which implements SAML protocols [4]. The base server functionality is exposed by the SAML Attribute Query Protocol. The SAML name identifier mapping protocol and SAML authentication request protocol are also implemented. In all cases the SOAP binding [11] is used. For the SAML authentication protocol the HTTP POST binding is also implemented. The server implements the core SAML specification as well as additional profiles to ensure a greater interoperability level:

– XACML Attribute Profile [12].
– SAML Attribute Query Deployment Profile for X.509 Subjects [13],

– SAML Attribute Self-Query Deployment Profile for X.509 Subjects [13],
– OGSA Attribute Exchange Profile Version 1.2 [14],

Our effort to be compliant to SAML related standards provides a solid foundation for an interoperability of the grid authorization systems. One of the largest projects in this field is the IVOM (Interoperability and Integration of VO-Management), a part of the D-Grid initiative. The IVOM aims to develop services that enable integration of VOMS and Shibboleth-based VO management systems with the grid middlewares used in Germany, in particular gLite, Globus Toolkit 4 and UNICORE 5. As it is suggested in [15], development of SAML enabled components for the grid middlewares can be seen as the most promising approach to enabling cooperation of the various authorization systems.

An implementation of the basic features — attribute, identities and group management — is obvious and won't be discussed here. However there are some features unique to the UVOS. The server keeps a whole history of performed operations and a past VOs state. Thanks to it it is possible to (a) check a whole content of the UVOS server as was used in any point in the past and (b) get the list of all events which occurred at the specified time frame. The events mechanism is also connected with an another UVOS functionality: notifications. Administrator can define an arbitrary number of notifications to be issued in a case of any modification operations. Currently only an e-mail notification is supported but other messaging backends can be added.

Finally, the UVOS web server is truly extensible by means of classic Java web applications (servlets), which can be simply installed by copying them into a designated server's installation directory. Two such web applications extending server's functionality are provided as ready to be used modules: one is providing authentication form for the SAML authentication request protocol, the second one supports enrollment of new users.

2.1 The Client Side

There are two management clients currently available: the UVOS command line client (UVOS CLC) and UVOS VO Manager. The command line client can be used to administer UVOS from the console in an interactive or batch mode. The UVOS VO Manager is a powerful GUI application based on the Eclipse Rich Client platform. It is much easier to use then a command line client, offers an intuitive interface so usually UVOS VO Manager a preferred choice.

Finally there is a number of UVOS consumers available. We present here consumers created in the Chemomentum project, but note that in principle any other SAML 2.0 attribute consumer may be used with the UVOS server.

The most important consumer is the UVOS module built into (and distributed with) UNICORE 6 server. This module is highly configurable and allows for gathering authorization data and to use selected attributes as a local UNIX account names. Thanks to the UVOS concept of attribute scopes, it is possible to keep mappings for multiple sites in a one place.

In addition to the UNICORE consumer there are currently available Globus 4.0 and 4.1+ consumers (there are two modules as both versions of the Globus Toolkit use different internal authorization API) so called *Policy Information Point* and *PolicyEnforcment Point*. They have similar functionality to the UNICORE consumer. There is also a consumer for web applications. It can use the UVOS to transparently authenticate web browser users. More information on this topic is presented in the section 4.

All basic components which were discussed in this section are presented in the figure 1.

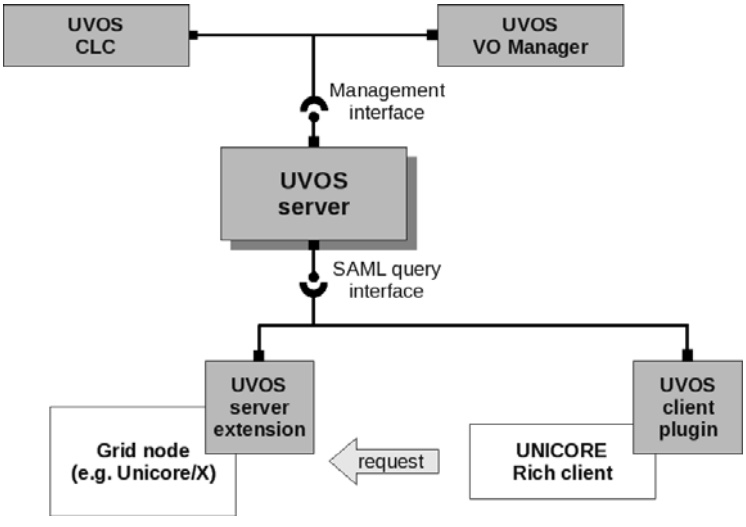


Fig. 1. Main components of the UVOS system along with their interconnections. In this figure the standard deployment in UNICORE grid middleware is presented.

3 VO Model

UVOS organizes entities within a hierarchical group structure. Top level groups of this structure are called virtual organizations, however they are no different then other groups. Each entity can be a member of arbitrary number of groups. It may have assigned a set of attributes; an attribute is composed of a name and a set of values, which can be empty. In addition, a single entity can possess multiple representations, for example in different formats. These equivalent incarnations of the same entity are called identities, and are usually invisible for an outside user.

The group membership is inherited in UVOS. The member of the subgroup becomes automatically the member of the parent group. This is different then e.g. in VOMS.

Every entity has a unique label and one or more tokens that represent it. The tokens must be in one of the supported formats, which currently are:

- a full X.509 certificate,
- an X.500 distinguished name,
- an e-mail address with an password used for authentication.

A token along with it's type is called an identity. The entity typically possesses one identity, but it can also have more. This reflects the real life situation where the single user can possess multiple certificates and email accounts. Additional identity formats may be added to the UVOS system with a intermediate level of effort. It is worth pointing out that all identities that compose an entity share the same characteristics (attributes, group membership, permissions, etc.): the UVOS works using entities internally.

3.1 Attributes

Attributes are composed of a name and a list of values. A name is a URI, and values are arbitrary strings. The value list can be empty. UVOS allows for three different ways of attributes assignment:

- using global attributes: an entity can have an attribute assigned globally. Such an attribute is valid always and in every context,
- using group-assigned attributes: an attribute can be assigned to a group, in which case all members of this group automatically hold this attribute (no matter if they were added later or prior to the creation of the group-assigned attribute). It is worth pointing out that this attribute is valid only in the scope of this group,
- using group-scoped entity attributes: those attributes are assigned to the entity, just like global attributes, but have an additional group restriction and are valid only in in the scope of the group.

The last two methods introduce a "group-scoped validity" of attributes, which requires a further explanation. The requester can ask (using the API provided by the UVOS service) for the entity's attribute either globally or valid only in a specified group. Global query returns global attributes only. A query limited to a group will return all entity's global attributes and all group-scoped attributes valid within the specified group.

For an even greater flexibility UVOS provides group inheritance of the scoped attributes. The group scoped attribute valid in a subgroup is also valid in its parent group. To illustrate an application of this feature let's consider group `/MainVO` and its subgroup `/MainVO/Admins`. In such scenario all attributes of the VO administrators who are the members of the later group are also valid for them in the main VO/group `/MainVO`.

UVOS allows for disabling temporary an attribute without deleting it. It is useful for instance to revoke some privileges for a period of time.

3.2 UVOS Authorization

UVOS access is restricted by it's own authorization stack. No external components/services are used to perform authorization. The authorization mechanism is advanced and probably the most advanced part of the whole UVOS system. UVOS supports a lot of authorization related features:

- full remote administration
- as a single server may carry multiple VOs it must be possible to assign management permissions for selected VOs/groups only
- the user should get additional permissions when accessing data about itself (e.g. to be able to change a password)

The general authorization mechanism used in UVOS is described in detail in its documentation. As it is quite extensive we have prepared a simple set of rules (which are used in a default UVOS configuration) that allow for an easy configuration of a secure UVOS access. The fundamental idea is to use a separate, special attribute that grants UVOS access permissions (and is not used for external purposes) with a fixed authorization policy for all groups. By assigning this attribute to VO members, the administrators can control privileges. Please note that this allows for taking the advantage of all types of attribute granting.

4 Deployments

There are several typical deployments in which UVOS can be used.

In the the so called *pull mode* a service (e.g. UNICORE server) contacts the UVOS server to obtain the attributes of a user who tries to use it. The attributes received from the UVOS server can be used for an authorization (e.g. server's policy may permit only those users who are in a certain UVOS group or who possess some attributes). Service may use received attributes to map requester to a local UNIX account. Pull mode is transparent for the grid users. However is more difficult for grid administrators to set it up: every grid site must be correctly configured to use UVOS.

In the *push mode* user has to contact a UVOS server on her own and get the list of possessed attributes in a signed assertion. Later this assertion can be attached to the requests which are sent to the grid services. If the service trusts the assertion issuer (i.e. the UVOS server which issued it) then it can use the attributes for authorization. Note that user can ask the UVOS server for a subset of owned attributes. In such case user can hide part of her/his identity or alter the execution (e.g. by choosing her role). The pull mode is more scalable in terms of server administration and easier to set up. However it requires user interaction and thus is more suitable for advanced grid users. A problem with expired assertions occurs here.

UVOS can be used to authenticate web browser users. SAML 2.0 authentication protocol is used to achieve it in a secure way. The UVOS server provides an implementation of the required SAML protocol. To make whole process operable

a web login interface is required. Appropriate module is available in the UVOS suite and can be installed as an extension of the core UVOS server. The details of the authentication process can be read in appropriate SAML 2.0 specification documents [11,12]. Here we only summarize that UVOS uses the SAML 2.0 SSO authentication profile with the HTTP POST binding.

Web developers can easily take advantage of the authentication pattern described above as software modules supporting it have been created. Currently the modules for authentication in Tomcat 6 containers are available.

5 UVOS from the Users Perspective

UVOS in the grid environment may be nearly transparent for the users when the pull mode is used. In this case only initial registration is required (if not performed manually by the grid administrator). Things get a little bit more complicated when the push mode or access via web portal is performed.

UVOS provides support for collecting and processing VO registration requests (called *VO applications*) from the users interested to join VO. The registration subsystem is flexible and offers many features. It is organized as follows:

- The *VO application form* is used to specify overall rules that applications must obey. It also contains additional information about data presented to the applying user. Examples of included information are: VO agreement, the group to which the application is connected etc. The VO application form is defined by the administrator and the UVOS server stores it. Multiple forms may be stored simultaneously.
- The VO application form is made accessible via the web interface which renders it.
- The *VO application* is issued by the user who files out the form using a web browser. The application is stored in the UVOS server database.
- The VO form administrator processes and accepts or rejects the application.

The UVOS server provides all necessary facilities to deploy the registration process except a WWW rendering of the VO application forms. To enable this feature a special extension must be installed, which is available in UVOS suite and installed in a similar manner as extension which supports the SAML authentication.

The VO application forms must be defined using the UVOS command line client (it is impossible to do this using the UVOS VO Manager). The specification of the application form is done in XML format. A VO application form can define its description, agreement, the linked group, identity types which are accepted by the form and much more.

To process VO applications, the VO administrator can use either UVOS CLC or UVOS VO Manager. It is suggested to use the latter one as it provides a full featured interface where all the details of the application can be reviewed and modified before committing. When the application form is accepted, a new identity is added to the UVOS database. It becomes a member of the group linked

to the application form. Additionally extra attributes (if such were provided) may be assigned to the new identity.

In *push* mode deployments it is the user's responsibility to select and provide a set of attributes which shall be used for authorization. A plugin for the UNICORE Rich client was created with the required functionality. It basically provides two features: simple browsing of all attributes which are defined for the user and creation of assertions to be used in *push* mode.

Finally there was created an application (employing Java Web Start technology) which allows for requesting an email account (typically users first registers with their certificate, by using VO registration facilities described above). On the server side it uses the same infrastructure as is used for the standard VO applications submitted via WWW form. After acceptance user can log to the portal without a need to load into a browser a certificate and a corresponding private key.

6 Conclusions

UVOS is a powerful VO management solution. It provides a low-level infrastructure with many advanced and useful features so its adoption is very fruitful for the grid. This was observed in Chemomentum project testbed grid where UVOS proved its value. Despite its complexity we managed to keep the system *easy*. For instance demo installation of the UVOS server takes around 5 minutes, and making it production ready requires, in most cases, to change demo server's certificate to a correct one. With an integrated support for the UNICORE middleware, components for the web systems and (currently being developed) Globus support modules, the UVOS can be seen as a good candidate for an interoperable and advanced VO founding software. We can stress here that during many months of testing in the Chemomentum testbed the UVOS system proved to be very stable. Only few minor bugs/problems were reported, and all were immediately fixed.

Deployment of UVOS does not mean that performance is scarified. Performance evaluation of the UVOS engine (i.e. without taking into account the network communication) as performed on the average hardware (Intel Core 2 Duo 3.16 GHz) with the embedded database, showed that UVOS can handle more then 100 operations per second (for all operations except of the removal of the group with complicated contents which is slightly slower). Most of the typical read operations are performed at the speed varying from 300 up to 2000 operations per second. Those tests are obviously not very detailed but shows that UVOS performance will not be a problem even in case of a large number of users.

While UVOS system is mostly complete, we are still working on improving it. The main effort currently is targeted at providing an easy to be used Globus support. Currently all Globus 4 OGSA compatible services can be authorized with the UVOS server. We plan to perform more interoperability tests with other SAML solutions. Those tests will involve not only the server but also the client side.

The UVOS releases can be downloaded from the UNICORE project web site [6]. The UVOS possess its own web site [16] where additional information can be found. This work was supported by European Commission under IST grant Chemomentum (No. 033437).

References

1. Alfieri, R., et al.: From gridmap-file to VOMS: managing authorization in a Grid environment. Future Generation Computer Systems (FGCS) 21(4), 549–558 (2005)
2. Welch, V., Barton, T., Keahey, K., Siebenlist, F.: Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration. In: 4th Annual PKI R&D Workshop (2005), http://grid.ncsa.uiuc.edu/papers/gridshib-pki05-final.pdf (2009)
3. The Shibboleth project (2009), http://shibboleth.internet2.edu
4. Cantor, S., et al. (eds.): Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS Standard (March 15, 2005), http://docs.oasis-open.org/security/saml/v2.0/ (2009)
5. The Chemomentum project (2009), http://www.chemomentum.org
6. The UNICORE project (March 2009), http://www.unicore.org
7. Foster, I., et al. (eds.): The Open Grid Services Architecture, Version 1.5. Open Grid Forum (July 24, 2006)
8. The iBATIS project (2009), http://ibatis.apache.org
9. The HSQL DB project (2009), http://hsqldb.org
10. Snelling, D., van den Berge, S., Li, V.: Explicit Trust Delegation: Security for Dynamic Grids. Fujitsu Scientific & Technical Journal (FSTJ), Special Issue on Grid Computing 40(2) (December 2004); Important note: this paper describes the initial ETD concept which was used in the UNICORE 5. Currently in the UNICORE 6 a highly extended version of the ETD approach is used, and to our knowledge there is no publication covering it yet
11. Cantor, S., et al. (eds.): Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS Standard (March 15, 2005), http://docs.oasis-open.org/security/saml/v2.0/ (2009)
12. Hughes, J., et al. (eds.): Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0, OASIS Standard, March 15 (2005), http://docs.oasis-open.org/security/saml/v2.0/ (2009)
13. Scavo, T. (ed.): SAML V2.0 Deployment Profiles for X.509 Subjects, OASIS Committee Specification (March 27, 2008), http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml2-profiles-deploy-x509-cs-01.pdf (2009)
14. Venturi, V., Scavo, T., Chadwick, D.: OGSA Attribute Exchange Profile Version 1.2. Open Grid Forum (2007)
15. Groeper, R., et al.: A concept for attribute-based authorization on D-Grid resources. Future Generation Comp. Syst. 24(3) (2009)
16. The UVOS project (March 2009), http://uvos.chemomentum.org

Application of ADMIRE Data Mining and Integration Technologies in Environmental Scenarios

Marek Ciglan[1], Ondrej Habala[1], Viet Tran[1], Ladislav Hluchy[1], Martin Kremler[2], and Martin Gera[2]

[1] Institute of Informatics of the Slovak Academy of Sciences, Dubravska cesta 9, 84507 Bratislava, Slovakia
[2] Comenius University, Faculty of Mathematics Physics and Informatics, Mlynska dolina, 84248 Bratislava, Slovakia

Abstract. In this paper we present our work on the engine for integration of environmental data. We present a suite of selected environmental scenarios, which are integrated into a novel data mining and integration environment, being developed in the project ADMIRE . The scenarios have been chosen for their suitability for data mining by environmental experts. They deal with meteorological and hydrological problems, and apply the chosen solutions to pilot areas within Slovakia. The main challenge is that the environmental data required by scenarios are maintained and provided by different organizations and are often in different formats. We present our approach to the specification and execution of data integration tasks, which deals with the distributed nature and heterogeneity of required data resources.

Keywords: Environmental applications, distributed data management, data integration, OGSA DAI.

1 Introduction

We present our work in the project ADMIRE[1], where we use advanced data mining and data integration technologies to run an environmental application, which uses data mining instead of standard physical modeling to perform experiments and obtain environmental predictions. The paper starts with description of the project ADMIRE, its vision and goals. Then we describe the history and current status of the environmental application. The core of the paper then presents our approach to the integration of data from distributed resources. We have developed a prototype of data integration engine that allows users to specify data integration process in form of a workflow of reusable processing elements.

[1] This work is supported by projects ADMIRE FP7-215024, APVV DO7RP-0006-08, DMM VMSP-P-0048-09, SEMCO-WS APVV-0391-06, VEGA No. 2/0211/09.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 165–173, 2010.

1.1 The EU ICT Project ADMIRE

The project ADMIRE (Advanced Data Mining and Integration Research for Europe [1]) is a 7th FP EU ICT project aims to deliver a consistent and easy-to-use technology for extracting information and knowledge from distributed data sources. The project is motivated by the difficulty of extracting meaningful information by mining combinations of data from multiple heterogeneous and distributed resources. It will also provide an abstract view of data mining and integration, which will give users and developers the power to cope with complexity and heterogeneity of services, data and processes. One of main goals of the project is to develop a language that serves as a canonical representation of the data integration and mining processes.

1.2 Flood Forecasting Simulation Cascade

The Flood Forecasting Simulation Cascade is a SOA-based environmental application, developed within several past FP5 and FP6 projects [2], [3], [4]. The application's development started in 1999 in the 5th FP project ANFAS [5]. In ANFAS, it was mainly one hydraulic model (the FESWMS [6]). It then continued with a more complex scenario in 5th FP project CrossGrid, turned SOA in 6th FP projects K-Wf Grid and MEDIgRID, and finally extended the domain to environmental risk management in ADMIRE. The application is now comprised of a set of environmental scenarios, with the necessary data and code to deploy and execute them. The scenarios have been chosen and prepared in cooperation with leading hydro-meteorological experts in Slovakia, working mainly for the Slovak Hydrometeorological Institute (SHMI), Slovak Water Enterprise (SWE), and the Institute of Hydrology of the Slovak Academy of Sciences (IH SAS). We have gathered also other scenarios from other sources, but in the end decided to use the ones presented below, because they promise to be the source of new information for both the environmental domain community, as well as for the data mining community in ADMIRE. Together with the scenarios, we have gathered a substantial amount of historical data. SWE has provided 10 years of historical data containing the discharge, water temperature, and other parameters of the Vah Cascade of waterworks (15 waterworks installations and reservoirs in the west of Slovakia). SHMI has provided 9 years of basic meteorological data (precipitation, temperature, wind) computed by a meteorological model and stored in a set of GRIB (Gridded Binary) files, hydrological data for one of the scenarios, and also partial historical record from their nation-wide network of meteorological data. They have also provided several years of stored weather radar data, necessary for one of the scenarios. The programs used by the application are in the context of ADMIRE described in Data Mining and Integration Lanuage (DMIL) [7]. The processes described in DMIL perform data extraction, transformation, integration, cleaning and checking. Additionally, in some scenarios we try to predict future values of some hydro-meteorological variables; if necessary, we use a standard meteorological model to predict weather data for these cases.

2 Environmental Scenarios of ADMIRE

In this chapter we present the suite of environmental scenarios, which we use to test the data mining and integration capabilities of the ADMIRE system. The scenarios are part of the Flood Forecasting and Simulation Cascade application, which has been in the meantime expanded beyond the borders of flood prediction into a broader environmental domain. There are four scenarios, which are in the process of being implemented and deployed in the ADMIRE testbed. These scenarios have been selected from more than a dozen of candidates provided by hydro-meteorological, water management, and pedological experts in Slovakia. The main criterion for their selection was their suitability for data mining application. The scenarios are named ORAVA, RADAR, SVP, and O3, and they are in different stages of completion, with ORAVA being the most mature one, and O3 only in the beginning stages of its design.

2.1 ORAVA

The scenario named ORAVA has been defined by the Hydrological Service division of the Slovak Hydrometeorological Institute, Bratislava, Slovakia. Its goal is to predict the water discharge wave and temperature propagation below the Orava reservoir, one of the largest water reservoirs in Slovakia.

The pilot area covered by the scenario (see Figure 1) lies in the north of Slovakia, and covers a relatively small area, well suitable for the properties of testing ADMIRE technology in a scientifically interesting, but not too difficult setting.

The data, which has been selected for data mining, and which we expect to influence the scenario's target variables - the discharge wave propagation, and temperature propagation in the outflow from the reservoir to river Orava - is depicted in Table 1.

For predictors in this scenario, we have selected rainfall and air temperature, the discharge volume of the Orava reservoir and the temperature of water in the Orava reservoir. Our target variables are the water height and water temperature measured at a hydrological station below the reservoir. As can be seen in Figure 1, the station directly below the reservoir is no.5830, followed by 5848 and 5880. If we run the data mining process in time T, we can expect to have at hand all data from sensors up to this time (first three data lines in Table 1). Future rainfall and temperature can be obtained by running a standard meteorological model. Future discharge of the reservoir is given in the manipulation schedule of the reservoir. The actual data mining targets are the X and Y variables for times after time T (T being current time).

2.2 RADAR

This experimental scenario tries to predict the movement of moisture in the air from a series of radar images (see for example). Weather radar measures the reflective properties of air, which are transformed to potential precipitation before being used for data mining. An example of already processed radar sample

Table 1. Depiction of the predictors and variables of the ORAVA scenario

Time	Rainfall	$Temp_Air$	Discharge	$Temp_Reservoir$	$Height_Station$	$Temp_Station$
T-2	$R_T - 2$	$F_T - 2$	$D_T - 2$	$E_T - 2$	$X_T - 2$	$Y_T - 2$
T-1	$R_T - 1$	$F_T - 1$	$D_T - 1$	$E_T - 1$	$X_T - 1$	$Y_T - 1$
T	R_T	F_T	D_T	E_T	X_T	Y_T
T+1	$R_T + 1$	$F_T + 1$	$D_T + 1$	$E_T + 1$	$X_T + 1$	$Y_T + 1$
T+2	$R_T + 2$	$F_T + 2$	$D_T + 2$	$E_T + 2$	$X_T + 2$	$Y_T + 2$

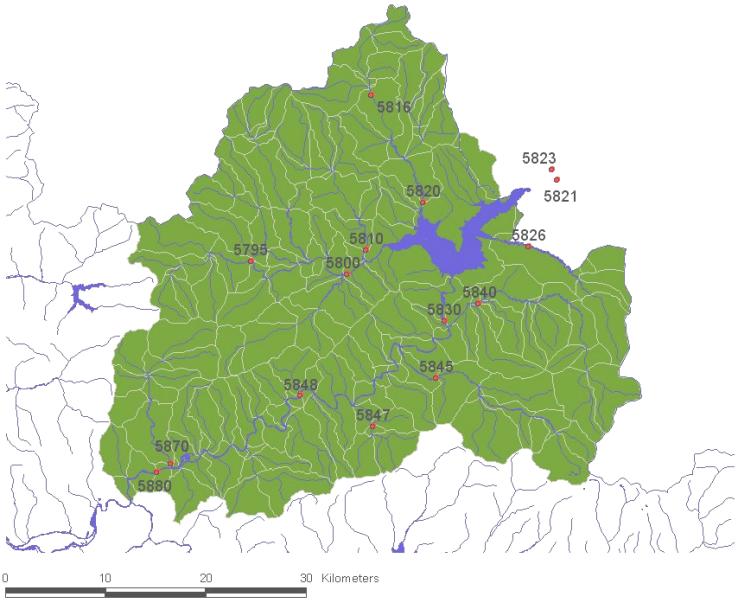


Fig. 1. The geographical area of the pilot scenario ORAVA

(with the reflection already re-computed to millimeters of rainfall accumulated in an hour) can be seen in Figure 2.

The scenario once again uses both historical precipitation data (measured by sensors maintained by SHMI) and weather predictions computed by a meteorological model. Additionally to these, SHMI has provided several years' worth of weather radar data (already transformed to potential precipitation).

2.3 SVP

This scenario, which is still in the design phase, is the most complex of all scenarios expected to be deployed in the context of ADMIRE. It uses the statistical approach to do what the FFSC application did before ADMIRE - predict floods. The reasons why we decided to perform this experiments are mainly the complexity of simulation of floods by physical models when taking into account more of the relevant variables, and the graceful degradation of results of the data

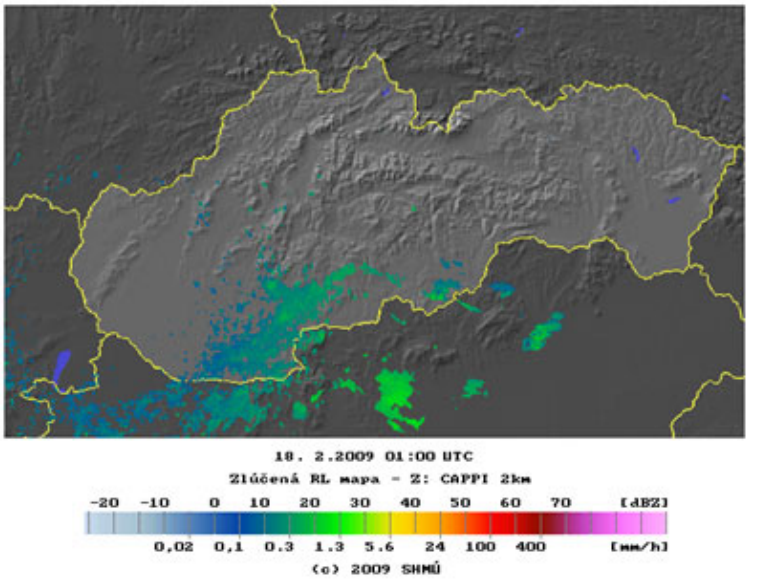


Fig. 2. An example of weather radar image with potential precipitation

mining approach when facing incomplete data - in contrast to the physical modeling approach, which usually cannot be even tried without having all the necessary data.

For predicting floods, we have been equipped with 10 years of historical data from the Vah cascade of waterworks by the Slovak Water Enterprise, 9 years of meteorological data (precipitation, temperature, wind) computed by the ALADIN model at SHMI, hydrological data from the river Vah, again by SHMI, and additionally with measured soil capacity for water retention, courtesy of our partner Institute of Hydrology of the Slovak Academy of Sciences. We base our efforts on the theory, that the amount of precipitation, which actually reaches the river basin and contributes to the water level of the river is influenced by actual precipitation and its short-term history, water retention capacity of the soil, and to lesser extent by the evapotranspiration effect.

3 Data Integration Engine for Environmental Data

In this section, we discuss the data integration engine designed for the environmental data integration and mining. It is motivated by the scenarios described in previous section. We first describe requirements that we took into account and then we present our approach to environmental data integration. In the discussion, we give examples mainly from Orava scenario; the first scenario implemented using our data integration engine.

In Orava river management scenario, the data from three different sources are used. The data are owned and maintained by different organizations. To allow the

data mining operations proposed for this scenario, the data from those different sources must be integrated first. Furthermore, the data are kept in different formats. In the case of Orava scenario, two data sets are stored in relational database (waterworks data, water stations measurements) and one is kept in binary files (precipitation data are stored in GRIB files - binary file format for meteorological data). From technical point of view, we must be able to work with the heterogeneous data stored in distributed, autonomous resources. In our work, we have considered so far the data in the form of lists of tuples.

In the following, we use the term data resource to denote a service providing access to data, with a single point of interaction. We use the term processing resource to denote a service capable of performing operations on the input lists of tuples. Data resource can have capabilities of a processing resource.

Atomic units used for data access and transformations are called processing elements (PE). Following types of processing elements are needed:

- Data retrieval PEs - operations able to retrieve the data from different, heterogeneous data sources. Data retrieval PEs are executed at data resources. This class of PEs is also responsible for transforming raw data sets to the form of tuples.
- Data transfer PE - able to transfer list of tuples between distinct processing resources.
- Data transformation PE - operations that transform input list of tuples. These PEs can perform data transformation on per tuple basis, or can be used to aggregate tuples in the input lists.
- Data integration PEs - given input lists of tuples, data integration operations combine the tuples from input lists into a coherent form.

An operation has one or more inputs and one or more outputs. Inputs can be either literals or list of tuples and a outputs are list of tuples. Operations can be chained to form a data integration workflow - an oriented graph, where nodes are operations and edges are connection of inputs and outputs of the operations.

The term Application Processing Element (APE) will denote a data integration workflow that can be executed at a single resource. APE is a composition of atomic operations that provides functionality required by a data integration task. For example, in Orava scenario we use the precipitation data from GRIB files. The GRIB reader processing element extracts the data from GRIB files; it has two inputs - the first is a list of GRIB files and the second is a list of indexes in GRIB value arrays. The GRIB reader activity outputs all the values at input indexes from all the input files. We use an operation that queries the GRIB metadata database to determinate GRIB files of interest and another operation that transform given geo-coordinates in WGS84 to the indexes consumed by GRIB reader activity. This small workflow of three operations forms a single APE that provides precipitation data for given time period and geo-coordinates. The idea behind APE is to provide data integration blocks that can be executed at a single processing or data resource and can be reused for in multiple data integration tasks. Similarly to atomic PE, the inputs of APE can be literals or list of tuples and outputs are list of tuples.

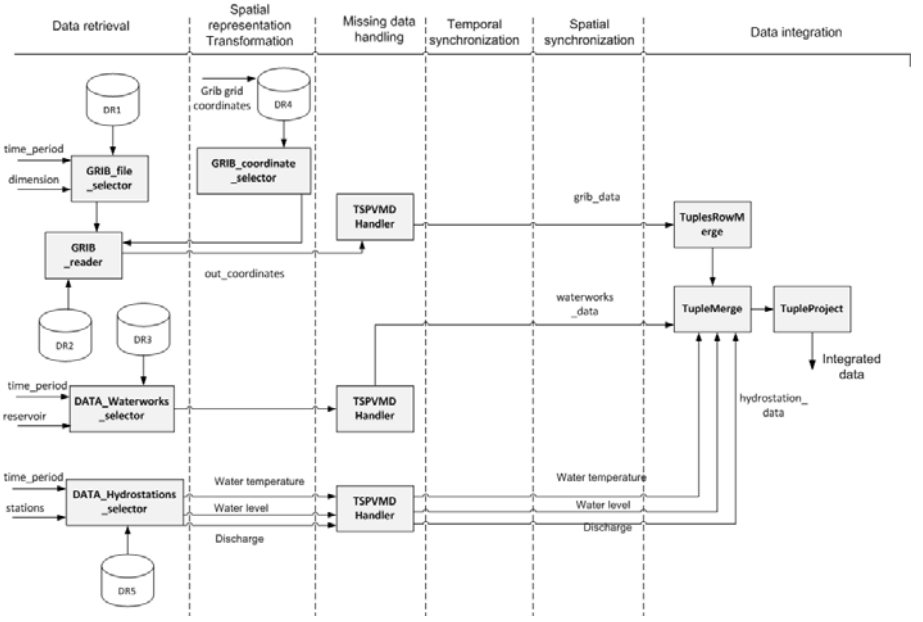


Fig. 3. Orava river management scenario - APEs workflow

The goal of our proposed data integration engine is to provide means of executing data integration tasks that are composed of multiple APEs and can integrate the data from distributed, autonomous and possibly heterogeneous data resources. Our data integration engine is designed to run the data integration tasks, given the input parameters and the APE workflow specification.

APE workflow specification is composed of four components: definition of APEs instances, mapping between inputs and outputs of connected APEs, mapping between the definition of integration task parameters and the parameter inputs of APEs in workflow and the definition of the result output.

In alignment with ADMIRE project vision, the APEs are specified in Data Mining and Integration Language (DMIL) [7] that is being developed within the project. The goal of DMIL is to be a canonical representation of data integration process, described in an implementation independent manner. The APE instance is specified by the DMIL description of the process that should be executed, the specification of the data/processing resource it should be executed at and APE instance identifier that is unique within the APE workflow specification. Figure 3 depicts the APEs workflow of the Orava river management scenario.

In our view, the main advantage of proposed data integration engine is that user can specify sub-workflows that are executed on a separate data resources and the engine automatically connects the results of APEs executed on distributed resources. This helps to deal with the complexity of the distributed data integration.

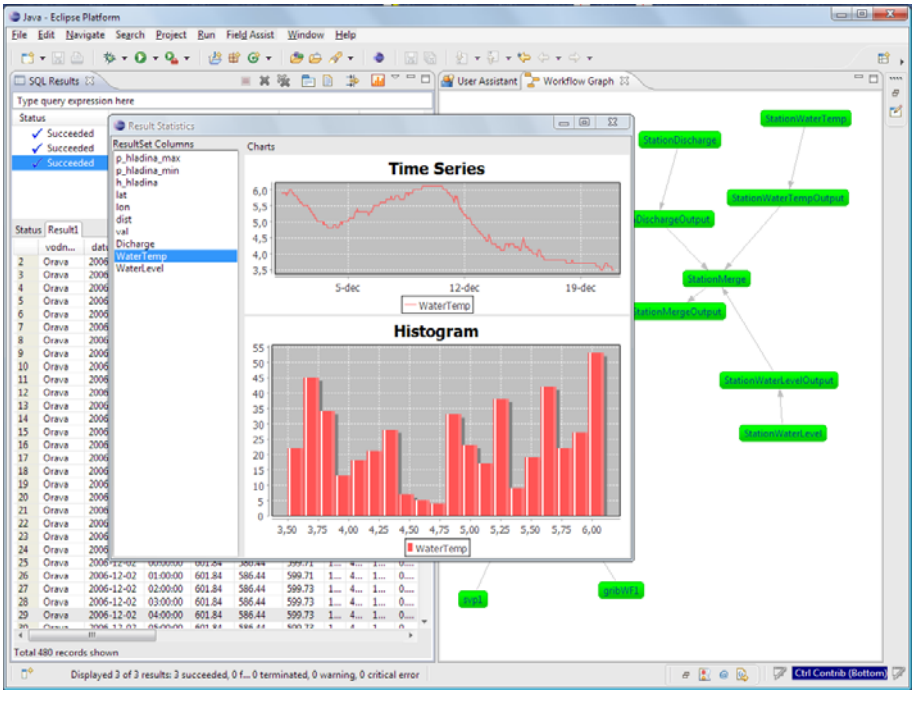


Fig. 4. GUI showing results of DIEED - APEs workflow and its results

3.1 Implementation

The prototype of proposed data integration engine for environmental data (DIEED) is implemented in JAVA programming language. It uses OGSA-DAI ([8], [9]) framework as the platform for exposing data resources in the distributed testbed and for executing the partial workflows of processing elements; it also provide us with the data transfer capabilities and streaming of the list of tuples between remote nodes. The data integration engine takes as inputs the integration task parameters and APE workflow specification. From the APE workflow specification, the engine constructs an oriented graph of APEs (defined by the mapping between inputs and outputs of APEs). For each node of the graph (containing an APE specified in DMIL) the DIEED performs following actions:

1. Compiles DMIL code - the DMIL specification of the node process is compiled to JAVA class that constructs an OGSA-DAI workflow.
2. JAVA class containing OGSA-DAI workflow is compiled by JAVA compiler, it is instantiated and OGSA-DAI workflow object is created
3. workflow object is submitted to OGSA-DAI service for execution
4. workflow execution on remote server is monitored

The whole APEs workflow is monitored during execution (providing information on the state of each of APEs); after execution is finished, the results can be retrieved in form of WebRowSet object.

DIE was integrated with the toolkit being developed in the project; this allows the user to submit APEs workflows, visualize the specified workflow and monitor its execution via graphical user interface based on Eclipse platform. Figure 4 depicts the graphical user interface for DIEED.

4 Conclusion

In this paper, we have presented preliminary results of our ongoing work on the data integration engine for environmental data that is being developed in the scope of ADMIRE project. We have first described four scenarios dealing with the integration and mining of environmental data. The main challenge is that the environmental data required by scenarios are maintained and provided by different organizations and are often in different formats. Our work concentrated on providing a platform that would allow integration of data from distributed, heterogeneous resources. Our results allow users to construct reusable application processing elements specified in DMIL [7] (language for data mining and integration, which is being designed within the project) and the engine executes them transparently on distributed data resources.

References

1. ADMIRE. EU FP7 ICT project: Advanced Data Mining and Integration Research for Europe (ADMIRE), 2008-2011. Grant agreement no. 215024, http://www.admire-project.eu (accessed November 2009)
2. CROSSGRID. EU FP5 IST RTD project: Development of Grid Environment for Interactive Applications (2002-05) IST-2001-32243, http://www.eu-crossgrid.org (accessed April 2009)
3. K-Wf Grid. EU FP6 RTD IST project: Knowledge-based Workflow System for Grid Applications (2004-2007) FP6-511385, call IST-2002-2.3.2.8, http://www.kwfgrid.eu (accessed August 2008)
4. MEDIGRID. EU FP6 RTD Sust. Dev. project: Mediterranean Grid of Multi-Risk Data and Models (2004-2006) GOCE-CT-2003-004044, call FP6-2003-Global-2
5. ANFAS. EU FP5 IST RTD project: datA fusioN for Flood Analysis and decision Support (2000-2003) IST-1999-11676
6. Finite Element Surface Water Modeling System (FESWMS), http://smig.usgs.gov/SMIC/model_pages/feswms.html (accessed November 2009)
7. Atkinson, M., et al.: ADMIRE White Paper: Motivation, Strategy, Overview and Impact, v0.9 (2009)
8. Antonioletti, M., Atkinson, M.P., Baxter, R., Borley, A., Chue Hong, N.P., Collins, B., Hardman, N., Hume, A., Knox, A., Jackson, M., Krause, A., Laws, S., Magowan, J., Paton, N.W., Pearson, D., Sugden, T., Watson, P., Westhead, M.: The Design and Implementation of Grid Database Services in OGSA-DAI. Concurrency and Computation: Practice and Experience 17(2-4), 357–376 (2005)
9. Karasavvas, K., Antonioletti, M., Atkinson, M.P., Chue Hong, N.P., Sugden, T., Hume, A.C., Jackson, M., Krause, A., Palansuriya, C.: Introduction to OGSA-DAI Services. In: Herrero, P., S. Pérez, M., Robles, V. (eds.) SAG 2004. LNCS, vol. 3458, pp. 1–12. Springer, Heidelberg (2005)

Performance Based Matchmaking on Grid

Andrea Clematis[1], Angelo Corana[2], Daniele D'Agostino[1], Antonella Galizia[1], and Alfonso Quarati[1]

[1] IMATI-CNR, via De Marini 6, 16149 Genova, Italy
{dagostino,clematis,galizia,quarati}@ge.imati.cnr.it
[2] IEIIT-CNR, via De Marini 6, 16149 Genova, Italy
corana@ieiit.cnr.it

Abstract. Grid Technologies supply users with high computational and storage resources to execute demanding applications. To this end, Grid environments must provide query and discovery tools, able to select the most suitable resource(s) satisfying application requirements. A description of application and resources, grounded on a common and shared basis, is therefore crucial to favour an effective pairing. A viable criterion to match demand (job) with supply (computational resource) is to characterize resources by means of their performance evaluated through benchmarks relevant to the application. We introduce GREEN, a distributed matchmaker, based on a two-level benchmarking methodology. GREEN facilitates the submission of jobs to the Grid, through the specification of both syntactic and performance requirements, independently of the underlying middleware and thus fostering Grid interoperability.

1 Introduction

One of the primary issues in Grid Computing is the "clever" discovery and selection of resources, so that a user could find quickly the resources he needs. Unfortunately, Grid middlewares offer basic services for the retrieving of information on single resources, and thus they are often inadequate to meet specific user requirements. A matchmaking component (e.g. broker, matchmaker) is responsible for carrying out this supply-demand coupling process [1].

We present a methodology to improve the matchmaking process based on information about performance of computational resources. Our aim is to integrate the information available via the Grid Information and Monitoring services by annotating resources with both low-level and application-specific performance metrics. These semantically relevant aspects of resources could be examined by a/the broker to filter out the solutions that best fit application requirements. A widespread method to measure and evaluate the performance of computer platforms is through benchmarking [2]. Application-specific benchmarks are widely acknowledged tools in the HPC domain, to measure the performance of resources stressing simultaneously several aspects of the system. Notwithstanding, so far application benchmarks have not been extensively considered on the Grid, due to diversified types of applications, architectural complexity, dynamic Grid behavior and heavy computational costs [3].

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 174–183, 2010.

On these bases, we developed GREEN (GRid Environment ENabler), a Grid service addressed both to Grid administrators and users. It assists administrators in the insertion of benchmark information related to every Physical Organization (PO) composing the Grid, and provides users with features which a) facilitate the submission of job execution requests, by specifying both syntactic and performance requirements on resources; b) support the automatic discovery and selection of the most appropriate resources. The aim of GREEN is the discovery of the resources that satisfy user requirements and their ordering by performance ranking. The selection phase is left to a (meta)scheduler, allowing to apply the preferred scheduling policies to meet specific purposes.

An important design goal of GREEN is interoperability. To this end, a unique standard language, JSDL [4], is used to express job submission requirements, and an internal translation to the job submission languages used by the various middlewares is performed. Middleware independence is pursued through an extension of JSDL based on the Glue2.0 schema [5]. Moreover, since we are interested in the execution of parallel applications, we borrowed from SPMD [6] some extensions to JSDL related to concurrency aspects.

The paper is organized as follow. Section 2 shortly describes related works; Sect. 3 discusses the main contributions in the job and resource characterization languages. Section 4 outlines the two-level benchmarking methodology. The description of design issues of GREEN and an analysis of the extensions operated to existing languages are reported in Sect. 5. Section 6 gives concluding remarks.

2 Related Works

The implementation of an efficient and automatic mechanism for the effective discovery of the resource that best suits a user job is one of the major problems in present Grids. The Globus toolkit does not provide a resource matchmaking/brokering as core service, but the GridWay metascheduler [7] was included as an optional high-level service since June 2007. GridWay allows users to specify only a fixed and limited set of resource requirements, most of them related to queue policies. This choice limits resource ranking, and benchmarks are not considered at all. On the contrary gLite has a native matchmaking/brokering service taking into account more requirements and benchmark values, although they are fixed and the service is based on a semi-centralized approach [8].

A way to improve the efficiency of resource discovery is to drive the search towards resources showing good performance in the execution of jobs with similar or known behavior. As explained in Sect. 4, the characterization of Grid resources based on pre-computed benchmarks seems a valid strategy to follow. The importance of benchmarking computational resources of Grids is largely acknowledged together with its criticality [9]. Actually, besides the set of interesting parameters to measure (e.g. CPU speed, memory size) different factors have to be taken into account when considering the execution of a benchmark on Grid. Several works proposed tools to manage and execute benchmarks on Grid. The Grid Assessment Probes [10] test and measure performance of basic Grid functions,

such as job submission, file transfers, and Grid Information Services (ISs). The GridBench tool [11] provides a graphical interface to define, execute and administrate benchmarks, also considering interconnection performance and resource workload. The NAS Grid Benchmark (NGB) suite [12] defines a set of computationally intensive benchmarks representative of scientific, post-processing and visualization workloads. A brokering mechanism based on benchmarking of Grid resources is proposed in [13]. However, the scope of the broker is focused on the ARC middleware and the NorduGrid and SweGrid production environments, and it adopts xRSL, an extension of RSL, to submit users jobs.

3 Resource and Job Characterization

To accomplish the matchmaking task, a proper description of resources is required both on owner and job/user side. To this end, different projects and research groups have proposed different languages.

On the resource-side, adequate information is required to advertise resource static (e.g. OS, number of processors) and dynamic (e.g. number of executing tasks, amount of free memory) properties. Actually, the main efforts in the direction of a standard resource-language come from the GLUE (Grid Laboratory Uniform Environment) Working Group, which deployed the Glue schema [5]. It is a conceptual model of Grid entities comprising a set of information specifications for Grid resources; an implementation through an XML Schema is given in [14]. As the schema has evolved, different versions have been used by various middlewares, leading to the Glue 2.0 specification. It foresees the benchmarking characterization of resources by specifying the *Benchmark_t* complex type referencing benchmarks of type defined by *BenchmarkType_t*. Through the latter, an open and extensible enumeration type, it is possible to specify a benchmark amongst a list of six values (e.g. specint2000, specfp2000, cint2006). Other values compatible with the string type and the recommended syntax are allowed.

On the client-side, a job submission request expressed via a Job Submission Languages (JSL), in addition to stating the application-related attributes (e.g. name and location of source code, input and output files), should express syntactic requirements (e.g. number of processors, main memory) and ranking preferences (if any) to guide and constraint the matching process on resources.

The Job Description Document (JDD) [15], introduced by Globus Alliance with the Web Services versions of the Globus Toolkit, defines an XML language closer to the XMLish dialects used in the WSRF family. The main purpose of a JDD document is to set the parameters for the correct execution of a job. The selection of the facilities to use has to be performed in advance by interacting with the WS MDS services of the available resources. In the JDD schema, it is possible to specify only few requirements, as the minimum amount of memory, or to set useful information as the expected maximum amount of CPU time. It is however possible to extend the schema with user-defined elements.

The Data Grid Project proposed the Job Description Language (JDL), afterwards adopted by the EGEE project [16]. A JDL document contains a flat

list of argument-value pairs, specifying two classes of job properties: job specific attributes and resources-related properties (e.g. *Requirements* and *Ranks*) used to guide the matching process towards the most appropriate resources. These values can be arbitrary expressions, which use fields published by resources in the MDS, and are not part of the predefined set of attributes for the JDL, as their naming and meaning depend on the adopted Information Service schema. In this way, JDL is independent of the resources information schema adopted.

The Job Submission Description Language (JSDL) developed by the JSDL-Working Group [4] of the Global Grid Forum, aims to synthesize consolidated and common features present in other JSLs, obtaining a standard language for the Grid. JSDL contains a vocabulary and normative XML Schema facilitating the declaration of job requirements as a set of XML elements. Likewise JDL, job attributes may be grouped in two classes. The *JobIdentification*, *Application* and *DataStaging* elements describe job-related properties. The *Resources* element lists some of the main attributes used to constraint the selection of the feasible resources (e.g. *CPUArchitecture*, *FileSystem*, *TotalCPUTime*). As only a rather reduced set of these elements is stated by the JSDL schema, an extension mechanism is foreseen. Examples of JSDL extension able to capture a more detailed description of the degree of parallelism of jobs are presented in [6,17].

4 A Two-Level Benchmarking Methodology

To describe Grid resources, we propose a two-level methodology aimed to give a useful enriched description of resources and to facilitate the matchmaking process. Our methodology considers two approaches: I) the use of micro-benchmarks to supply a basic description of resource performance; II) the deployment of application-driven benchmarks to get closer insight into the behavior of resources under more realistic conditions of a class of applications. Through application-driven benchmarks, it is possible to add an evaluation of the resources on the basis of the system indicators that are more stressed by an application. Our present aim is to provide a proper description of Grid resources in isolation, i.e. without considering complexity aspects of Grid environments. Future developments of this work would capture some of these aspects.

4.1 Micro-benchmarks

In order to supply a *basic* resource characterization, mainly based on low-level performance capacity, we consider the use of traditional micro-benchmarks. To this aim, a reasonable assumption is that the performance of a machine mainly depends on CPU, memory and cache, and interconnection performance [18]; therefore, we individuated a concise number of parameters to evaluate aimed to provide an easy-to-use description of the various nodes. Table 1 shows resource properties and related metrics measured by the employed micro-benchmarks. The micro-benchmarks used in this phase generally return many values. To obtain results usable in the matchmaking process, we considered for each benchmark synthetic parameters or the most significant value. They are used to characterize resources by populating the benchmark description managed by GREEN.

Table 1. Low-level benchmarks and related metrics

Resource capability	CPU	Memory	Memory-Cache	Interconnection	I/O
Metric	MFLOPS	MBps	MBps	MBps	MBps
Benchmark	Flops	Stream	CacheBench	Mpptest	Bonnie

4.2 Application-Specific Benchmarks

Micro benchmarks are a good solution when the user has little information about the job he is submitting, and for applications that are not frequently executed. Indeed, very often the participants to a Virtual Organization have similar aims, and therefore it is possible to identify a set of the most used applications. In these cases the most suitable approach is to evaluate system performance through application-specific benchmarks that approximate at best the real application workload. These benchmarks represent the second level of our methodology.

As case studies we considered applications of our interest, i.e. image processing, isosurface extraction, and linear algebra. For the first two classes, we choose a light version code aiming to emphasize precise aspects of the code. For image processing, we selected a compute intensive elaboration applied to a reference image of about 1 MB; in this way CPU metrics are mainly stressed. The isosurface extraction application provides a more exhaustive performance evaluation of the system, as it also heavily involves I/O operations. In this case, we considered the processing of a small 3D data set of 16 MB, producing a result of 67 MB. On the contrary, to represent the class of applications based on linear algebra, we used the well known Linpack benchmark [19]. The metric considered is execution time, the results are stored in the internal data structure of GREEN.

5 Benchmark-Driven Matchmaking

A huge gap separates users and resources, and tools that allow the two parts to better come to an agreement are highly useful. In [20] we presented GREEN, a Grid service based on a distributed and cooperative approach for Grid resource discovery. It supplies users with a structured view of resources (single machines, homogeneous and heterogeneous clusters) at the PO level, and leverages on an overlay network infrastructure which connects the various POs constituting a Grid. For each PO, a GREEN instance is deployed to keep updated information about the state of all PO's resources, and to exchange them with other GREEN instances in the discovery phase.

In this work, we describe an advanced version of GREEN able to characterize Grid resources through benchmark evaluations. Acting as a distributed matchmaker, GREEN manages and compares the enriched view of resources with user-submitted jobs, with the goal of selecting the most appropriate resources. Operating at intermediate level between applications (e.g. schedulers) and middleware, GREEN aims to discover the whole set of resources satisfying user requirements ordered by ranks. The selection of a particular resource

is left to a (meta)scheduler, to which the resources set is forwarded, to apply the preferred scheduling policies optimizing target functions (e.g. Grid throughput, QoS). Once the "best" resource is chosen, GREEN will be re-invoked to carry-out the submission of the job on it, via the Execution Environment (EE).

5.1 Benchmarking Grid Resources

GREEN supplies Grid administrators with the facility of submitting, executing benchmarks (both micro and application-related) against the resources belonging to a certain administrative domain (PO), and storing results.

To support the matching mechanism (i.e. the comparison with resources information contained in the previously acquired XML) the benchmark-value copies are directly represented as Glue entities according to the XML reference realizations of Glue 2.0. By employing the openness of *BenchmarkType_t* (as recalled in Sect. 3), the set of recognized benchmarks is extensible without any change to the document schema. An example of a benchmark document related to the execution of micro-benchmark Flops against the resource identified by the IP 150.145.8.160, resulting in 480 MFlops is:

```
<Benchmark>
   <LocalID>150.145.8.160</LocalID>
   <Type>MFlops</Type>
   <Value>480</Value>
   <BenchLevel>micro</BenchLevel>
</Benchmark>
```

Through the use of the extension mechanism defined in Glue specification, we enriched the *Benchmark_t* type by adding the element *BenchLevel* which specifies the benchmark level (by accepting two string values *micro* and *application*) according to our two-level methodology.

Once a benchmark is executed and its results collected, an XML fragment, similar to the one reported above, is created for each resource and inserted in an XML document (namely *Benchmark image*), managed by GREEN, which collects all benchmarks evaluation for the PO.

5.2 Extending JSDL

The counterpart of benchmarking resources is the ability for users submitting a job to express their preferences about the performance of target machines. As explained in Sect. 3, both JDD and JSDL do not provide any construct to this aim. We introduce an element *Rank* (of complex type *Rank_Type*) devoted to this task, which embeds a sub-element *BenchmarkType_t* corresponding to the one contained in our extension of the Glue Schema. In the context of JSDL, the *Value* sub-element (see list below) is to be intended as a threshold to be satisfied by the corresponding Value (related to the benchmark stated by *Type*) contained in the *Benchmark* element of any resource to be selected by the matchmaker.

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition
xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
xmlns:posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
xmlns:spmd="http://schemas.ogf.org/jsdl/2007/02/jsdl-spmd"
xmlns:rank="http://saturno.ima.ge.cnr.it/ima/jsdl/2009/01/jsdl-rank">
<jsdl:JobDescription>
  <jsdl:Application>
     <jsdl:ApplicationName>ParIsoExtrctn</jsdl:ApplicationName>
     <spmd:SPMDApplication>
       <posix:Executable>parisoextraction</posix:Executable>
       <posix:Argument> inputvolume.raw</posix:Argument>
       <posix:Argument>200</posix:Argument>
       <posix:Output>isosurface.raw</posix:Output>
       <spmd:NumberOfProcesses>4</spmd:NumberOfProcesses>
       <spmd:ProcessesPerHost>2</spmd:ProcessesPerHost>
       <spmd:SPMDVariation>http://www.ogf.org/jsdl/2007/02/
                      jsdl-spmd/MPICH2<spmd:SPMDVariation/>
     </spmd:SPMDApplication>
  </jsdl:Application>
  <jsdl:Resources>
     <jsdl:OperatingSystemType>
       <jsdl:OperatingSystemName>LINUX</jsdl:OperatingSystemName>
     </jsdl:OperatingSystemType>
     <rank:Rank>
       <rank:Type>IsoSurface_Benchmark</rank:Type>
       <rank:Value>300</rank:Value>
       <rank:BenchLevel>application</rank:BenchLevel>
     </rank:Rank>
  </jsdl:Resources>
</jsdl:JobDescription>
</jsdl:JobDefinition>
```

As we are interested in the execution of parallel applications, we borrowed from SPMD [6] an extension to JSDL that supports users with a description set of applications and resources related to concurrency aspects (e.g. number of processes, processes per host). An example of an extended JSDL document is presented, it contains the extensions related to parallel requirements, along with our extension to rank resources on benchmark specification. The document is requesting for nodes able to execute the application-level "IsoSurface_Benchmark" in no more than 300 time units. Note how the *Rank* element has been located inside the *Resource* one, according to the extension mechanism provided by JSDL schema.

5.3 Distributed Matchmaking Process

Figure 1 shows the main components of a GREEN instance along with some of their interactions with other middleware services, notably IS and EE, by considering a Grid composed of three POs. The Job Submission (JS) component receives requests of jobs submission initiated by users; depending on the activation mode it behaves just like a messages dispatcher or as a translator of JSL

documents, carrying out their subsequent submission to the EE. The Benchmark Evaluation (BE) supports administrators in the performance-based characterization of PO resources. The Resource Discovery (RD) is in charge of feeding GREEN with the state of Grid resources. RD operates both locally and globally by carrying out two tasks: 1) to discover the state of the PO resources; 2) to dispatch requests to other GREEN instances. As to the first task, RD dialogues with the underlying IS (e.g. MDS, gLite IS) that periodically reports the state of the PO in the form of an XML file conformed to the Glue version adopted by the underlying middleware. This document (namely the *PO snapshot*) is stored, as it is, and managed by GREEN to answer to external queries issued by various clients (e.g. other GREEN instances, meta-schedulers). To deal with different underlying middlewares transparently to Grid users and applications, the syntactic differences among the various versions of Glue are managed by GREEN through a conversion mapping at matching time. To accomplish the dispatching task, RD handles the so-called neighbors view. Depending on the number of POs, i.e. GREEN instances running, their management could consider different strategies, whose description is beyond the scope of the paper. The Matchmaker performs the matching among resources in the Grid, and their subsequent ranking, with the requirements expressed by the users through the application submission document. More in detail: a user submits an extended JSDL document through a Grid portal (1). The document is managed by the Resource Selector component, which initiates the distributed matchmaking by forwarding it to the JS component of a randomly selected GREEN instance (2) (e.g. PO2). JS activates the Matchmaker (3). This instance of matchmaker, namely the Master Matchmaker

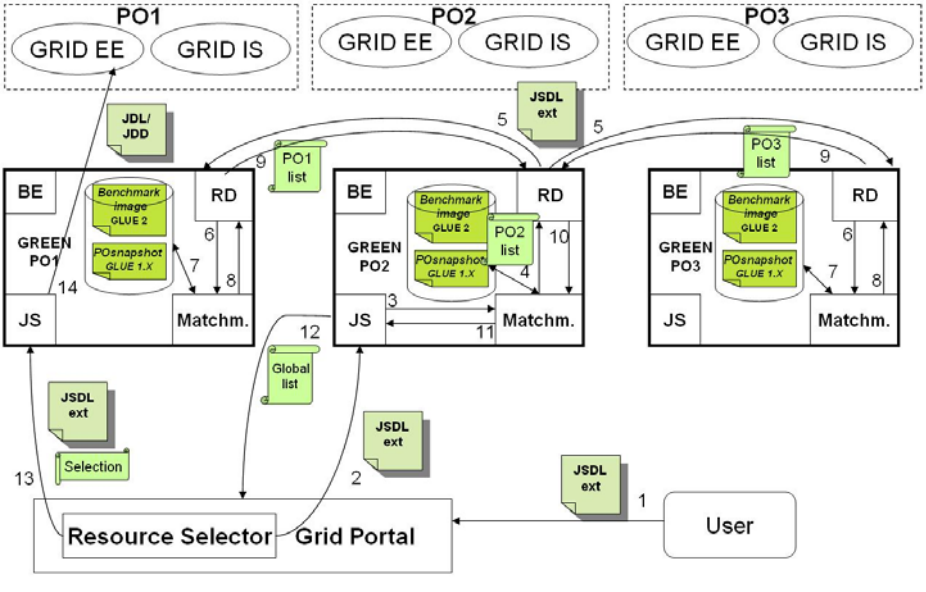


Fig. 1. Example of the matching phase with three GREEN instances

(MM), is responsible to provide the set of candidate resources to the Resource selector for this specific request. MM through RD forwards the document to all the other known GREEN instances and contemporaneously checks its local memory (4-5). All the matchmakers filter their *PO snapshot* selecting the set of PO resources satisfying the query. By analyzing the pre-computed *Benchmark image*, the satisfying resources with a *Value* element (for the chosen benchmark) that fulfils the threshold fixed in the corresponding *Rank* element of the JSDL document are extracted. The resources identifiers and their corresponding benchmark values are included in a list, called *PO list* which is returned to MM (6-10). MM merges these lists with its own PO list, producing a Global List ordered on the ranking values. The Global list is passed to JS (11) which returns it back to RS (12). Besides applying the selection policy to determine the resource to use, the Resource Selector calls the JS of the GREEN responsible of the PO owning the selected machine (GREEN PO1s instance in our case), by sending it the extended JSDL document along with the data identifying the selected resource (13). JS translates the information regarding the job execution of the original JSDL document in the format proper of the specific PO middleware, stating the resource on which the computation takes place. In particular, it will produce a JDD document for GT4 resources or a JDL document for the gLite ones. Finally, it activates the Execution Environment in charge of executing the job represented in the translated document (14).

6 Conclusions

To fill-in the gap separating users and resources, we designed GREEN, a distributed matchmaker providing Grid users with features to facilitate the submission of job execution requests containing performance requirements, in order to support the automatic discovery and selection of the most suitable resource(s). GREEN relies on a two-level benchmarking methodology: resources are characterized by means of their performance evaluated through the execution of low-level and application specific benchmarks. According to our methodology, every resource of a PO is tagged with the results obtained through the two levels of benchmarks and hence selectable, on performance basis, during the matchmaking phase. To ensure a good degree of independence from the underlying middlewares, GREEN leverages on two standards such as JSDL and Glue, that have been properly extended to take into account the performance-based description of resources.

References

1. Bai, X., Yu, H., Ji, Y., Marinescu, D.C.: Resource matching and a matchmaking service for an intelligent grid. International Journal of Computational Intelligence 1(3), 163–171 (2004)
2. Hockney, R.W.: The science of computer benchmarking. In: Software, environments, tools. SIAM, Philadelphia (1996)

3. Dikaiakos, M.D.: Grid benchmarking: vision, challenges, and current status. Concurrency and Computation - Practice & Experience 19(1), 89–105 (2007)
4. Anjomshoaa, A., Brisard, F., Drescher, M., Fellows, D., Ly, A., McGough, S., Pulsipher, D., Savva, A.: Job Submission Description Language (JSDL) Specification v1.0. Grid Forum Document GFD, 56 (2005)
5. Andreozzi, S.: GLUE Specification v. 2.0, rev. 3 (2009)
6. Savva, A. (ed.): JSDL SPMD Application Extension, Version 1.0. Grid Forum Document GFD.115, Open Grid Forum, OGF (2007)
7. Huedo, E., Montero, R.S., Llorente, I.M.: A Framework for Adaptive Scheduling and Execution on Grids. Software - Practice & Experience 34(7), 631–651 (2004)
8. gLite 3.1 User Guide, Doc. CERN-LCG-GDEIS-722398 (January 7, 2009), https://edms.cern.ch/file/722398/1.2/gLite-3-UserGuide.html
9. Nadeem, F., Prodan, R., Fahringer, T., Iosup, A.: Benchmarking Grid Applications for Performance and Scalability Predictions. In: CoreGRID Workshop on Middleware. Springer, Dresden (2007)
10. Chun, G., Dail, H., Casanova, H., Snavely, A.: Benchmark probes for grid assessment. In: 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), Santa Fe, New Mexico, USA. IEEE Computer Society, Los Alamitos (2004)
11. Tsouloupas, G., Dikaiakos, M.D.: GridBench: A Tool for the Interactive Performance Exploration of Grid Infrastructures. Journal of Parallel and Distributed Computing 67, 1029–1045 (2007)
12. Frumking, M., Van der Wijngaart, R.F.: NAS Grid Benchmarks: A tool for Grid space exploration. Cluster Computing 5(3), 315–324 (2002)
13. Elmroth, E., Tordsson, J.: Grid resource brokering algorithms enabling advance reservations and resource selection based on performance predictions. Future Generation Computer Systems 24(6), 585–593 (2008)
14. GLUE v. 2.0 Reference Realizations to Concrete Data Models (2008)
15. Job Description Document, http://www.globus.org/toolkit/docs/4.0/execution/wsgram/schemas/gram_job_description.html
16. Job Description Language, https://edms.cern.ch/file/555796/1/EGEE-JRA1-TEC-555796-JDL-Attributes-v0-8.pdf
17. Rodero, I., Guim, F., Corbal, J., Labarta, J.: How the JSDL can Exploit the Parallelism? In: Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2006), pp. 275–282 (2006)
18. Tsouloupas, G., Dikaiakos, M.: Characterization of Computational Grid Resources Using Low-level Benchmarks. In: Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing. IEEE Computer Society, Los Alamitos (2006)
19. Brent, R.: The LINPACK Benchmark on the AP 1000. Frontiers, pp. 128–135, McLean, VA (1992)
20. Clematis, A., Corana, A., D'Agostino, D., Gianuzzi, V., Merlo, A., Quarati, A.: A distributed approach for structured resource discovery on Grid. In: Int. Conference on Complex, Intelligent and Software Intensive Systems, Barcelona, pp. 117–125. IEEE Computer Society, Los Alamitos (2008)

Replica Management for National Data Storage

Renata Słota[1], Darin Nikolow[1], Marcin Kuta[1], Mariusz Kapanowski[1], Kornel Skałkowski[1], Marek Pogoda[2], and Jacek Kitowski[1,2]

[1] Institute of Computer Science, AGH-UST, al. Mickiewicza 30, 30-059, Kraków, Poland
[2] Academic Computer Center CYFRONET-AGH, ul. Nawojki 11, 30-950 Kraków, Poland
{darin,rena,kito}@agh.edu.pl

Abstract. National Data Storage is a distributed data storage system intended to provide high quality backup, archiving and data access services. These services guarantee high level of data protection as well as high performance of data storing and retrieval by using replication techniques. In this paper some conceptual and implementation details on creating a Prediction and Load Balancing Subsystem for replica management are presented. Preliminary real system test results are also shown.

1 Introduction

National Data Storage (NDS) is a distributed data storage system intended to provide high quality backup, archiving and data access services [1]. These services are capable of providing high level of data protection, data availability and data access performance. In order to guarantee these things replication techniques are used. Two problems arise with using this approach: selecting physical storage locations for newly created replicas and choosing the best replica for a given data transfer. If these problems are properly solved we can count on decreasing the access time to data. A side effect of using geographically distributed replicated data sets is also higher network and storage total throughput.

The client access to NDS is provided by Access Nodes (ANs). ANs spread over the country are located in national computer centers having direct links to the NDS Pionier backbone network [2]. The general idea is that client requests come via different ANs and the requested data is served by the most appropriate Storage Node (SN), selected separately for each request, being the one which can provide requested data fastest. In this way some natural load balancing is achieved depending on the client access pattern.

One of the tasks in the NDS project is to build a replica management subsystem with the high performance of data transfers in mind. This subsystem is called Prediction and Load Balancing Subsystem (PLBS). This paper presents some conceptual and implementation details on creating PLBS. Essential part of this research concerns replication and the development of replication policies, which should help achieving reasonable level of storage load balancing. These policies, described further, are based on the storage model mentioned above. Preliminary test results are also shown.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 184–193, 2010.

The rest of the paper is organized as follows: The next section presents the state of the art. The third section describes the Hierarchical Storage Management (HSM) model used in this research and how it is represented in the database tables. The forth section gives some overview of the replication strategies used in the system. The test results are presented in the fifth section and the last section concludes the paper.

2 State of the Art

In [3] five replication strategies for read only data are presented. The strategies have been tested using three different access patterns. The study assumes tiered network with a central data source. Similar network model having constant storage nodes locations in the network hierarchy is studied in [4]. Park et al. in [5] study replication with another network hierarchy with no central storage node, where the node distance is expressed as link bandwidth. Their technique might be better in the case when the Internet is used for data transfer.

The mentioned studies assume static hierarchy and do not take into account the dynamic changes of bandwidth and latency resulting from the load of distributed system. In [6] an attempt to cope with this problem has been made. For replica selection they propose a neural net based algorithm predicting the network transfer time of a replica. Another example of research on replica access time prediction based on previous data transfers measurements is [7] in which the Markov chains are used for prediction.

Authors of paper [8] propose 3 heuristic algorithms for selecting the location of new replica based on network latency parameters and number of client requests observed in a certain time interval from the past. Fair-Share replication presented in [9] for choosing new replica location takes into account previous access load of server as well as availability of storage device represented by their storage load. In this way better load balancing among the storage servers is achieved.

Tests of the proposed replication strategies in the studies mentioned by now are conducted by using simulations. Results of real implementation of proposed models and strategies using monitoring of existing storage environment are shown in [10] and in their previous studies. The presented in these papers replication algorithms embody, besides the data access cost imposed by the network, also the cost caused by storage devices capabilities. In the case when a distributed storage system uses high bandwidth network it turns out that the system bottleneck are the storage devices, which bandwidth can be additionally limited according to their actual access load.

The majority of replica selection algorithms assumes that many users access the same data sets. In the case of data storage service holding mainly private user data, users will rather access their own particular files (holding backups or archives). That is why, essential in this case is access load balancing increasing the overall system utilization and thus reducing the access cost. In the proposed solution essential part of the process of existing replica selection and the process of new replica location selection is focused on the evaluating of storage

system performance and evaluating of server load. The evaluation is based on the adopted Common Mass Storage System Model (CMSSM) proposed in [11].

3 PLBS Architecture

Prediction and Load Balancing System (PLBS) is responsible for load balancing of data access requests among the storage nodes being part of the NDS. PLBS consists of three subsystems (see Fig.1): adopted JMX Infrastructure Monitoring System (JIMS) [12], database for keeping the values of monitored parameters and Advanced Monitoring and Prediction Daemon (AMPD).

The JIMS based monitoring system consists of Monitoring Agents (JIMS MA) installed on every HSM system being part of NDS, JIMS Gateway collecting data from the agents and storing it to PLBS database. The AMPD is responsible for proposing the best replica and location according to the chosen replication policy (see section 5). One of the requirements for the AMPD is that it must quickly respond, so the user requesting a storage operation does not experience system or data unavailability. Monitoring parameters are measured cyclically by background threads and are stored in the database. In this way the actual parameters (for a certain time interval) can be quickly retrieved from the database and the AMPD can return the results.

The HSM monitoring parameters are derived from the CMSSM model proposed in [11]. The model specifies essential parameters of HSM systems which need to be monitored for later use by system performance prediction algorithms. Two types of parameters have been defined: static parameters changing their values rarely and dynamic parameters changing their values frequently. Part of the model used by the replication policies implemented in PLBS so far is presented below along with the database description.

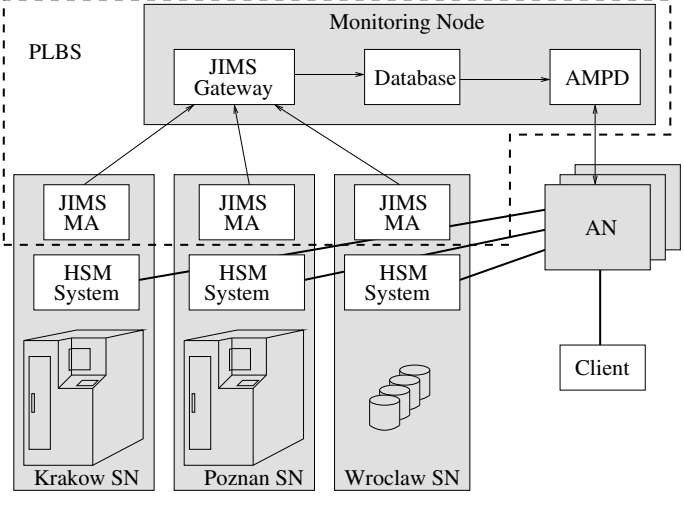


Fig. 1. The PLBS architecture

4 The PLBS Database

The goal of the PLBS database is to collect monitored parameters (from the CMSSM model) of distributed nodes in a single place. The approach to store current values of monitored parameters in a database has been chosen, because it allows to completely separate a application logic layer from a monitoring layer.

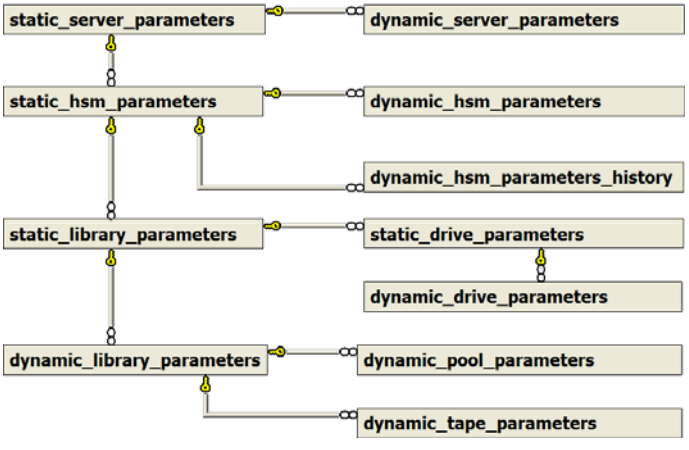


Fig. 2. The PLBS database diagram

The PLBS database is realized in the standard relational model and conforms to the CMSSM model. The structure of the database is derived from the structure of the monitored systems, which means that the database tables suit to essential HSM components, such as: libraries, drives, pools, tapes, disk cache, etc. The parameters, stored in the database, are divided into two groups: static parameters and dynamic parameters. A simple diagram showing relations between the PLBS database tables is shown in Fig. 2. The table columns specification is omitted for simplicity. The dynamic_hsm_parameters_history table stores history of changes of the dynamic HSM parameters. This table allows the application logic layer to make decisions based not only on the current values of parameters, but also on their statistical values over a certain time period.

Table 1 presents summary of the static parameters stored in the PLBS database, which are used in the replication policies. Updates of these parameters are performed only on user's demand, for example after a HSM system reconfiguration. Some of these parameters constitutes average values (like average disk cache transfer rate), which are provided by external measurements.

Table 2 presents summary of the dynamic parameters stored in the PLBS database, which are used in the NDS replication policies. These parameters are updated periodically. The update interval is set manually in the PLBS configuration files.

Table 1. Description of static parameters used in the replication policies

Parameter name	*Description*	*Implementation*
TotalCapacity	Estimated total capacity of a storage system installed on a single server.	The value of this parameter is estimated as a sum of disk cache capacities
TotalDCCapacity	Total capacity of a single HSM system disk cache.	The value of this parameter is received from the **df** UNIX systems command.
AverageDCReadRate	Estimated value of average disk cache read transfer rate.	The value of this parameter is measured by special benchmarks.
AverageDCWriteRate	Estimated value of average disk cache write transfer rate.	The value of this parameter is measured by special benchmarks.
NumberOfLibraries	Total number of tape libraries connected to a single server.	The value of this parameter is received from configuration files.

Table 2. Description of dynamic parameters, which are used in the replication policies

Parameter name	*Description*	*Implementation*
FreeCapacity	Estimated free capacity of a storage system installed on a single server.	The value of this parameter is estimated as a sum of free tapes capacity.
FreeDCCapacity	Free space in a single HSM system disk cache.	The value of this parameter is obtained from the **df** UNIX systems command.
CurrentRate	Transfer rate value from the last measurement.	The value of this parameter is measured by periodically.
HSMLoad	Number of requests waiting or being processed by the HSM system.	The value of this parameter is received from the **dsmq** command for the Tivoli Storage Manager (TSM) systems and from the **fsejob** command for the File System Extender (FSE) systems.

5 Replication Policies

The selection of SN for a given data access request is done by heuristic methods taking into account relevant monitoring parameters described in the previous section. Depending on the user profile an appropriate method (called further policy) is used. The AMPD component implements 4 replication policies: reading in shortest time - R_ST, reading from the minimally loaded device - R_ML, writing replicas of big files - W_BF, writing replicas to the minimally loaded device - W_ML. Each policy selects the location, for which the value *Loc*, defined in equations 1-4, is maximized.

The R_ST policy is defined by:

$$Loc = \alpha_1 \cdot \frac{RD}{RD_{Max}} + \alpha_2 \cdot \frac{CT}{RD} + \alpha_3 \cdot \frac{1}{1+HL}\,, \tag{1}$$

where RD – average disk cache read transfer rate, RD_{Max} – maximal value of average disk cache read transfer rate, taken over all locations, CT – current transfer rate, HL – hsm load, $\sum_{i\in\{1..3\}} \alpha_i = 1$, $\alpha_i > 0$. The exact meaning of these values is given in Tables 1 and 2.

Equation (2) expresses the R_ML policy:

$$Loc = \beta_1 \cdot \frac{ND}{ND_{Max}} + \beta_2 \cdot \frac{1}{1+HL} + \beta_3 \cdot \frac{1}{1+CL}\,, \tag{2}$$

where ND – number of drives, ND_{Max} – maximal value of number of drives, taken over all locations, CL – CPU load, $\sum_{i\in\{1..3\}} \beta_i = 1$, $\beta_i > 0$.

Each writing policy determines first whether enough free space is available in a HSM system. Equation 3 defines the W_BF policy.

$$Loc = \gamma_i \cdot \frac{FC_{DC}}{TC_{DC}} + \gamma_2 \cdot \frac{FC}{TC} + \gamma_3 \cdot \frac{WR}{WR_{Max}} + \gamma_4 \cdot \frac{1}{1+HL}\,, \tag{3}$$

where FC_{DC} – free disk cache capacity, TC_{DC} – total disk cache capacity, FC – free capacity, TC – total capacity, WR – average disk cache write transfer rate, WR_{Max} – maximal value of average disk cache write transfer rate, taken over all locations, $\sum_{i\in\{1..4\}} \gamma_i = 1$, $\gamma_i > 0$.

The policy W_ML is defined by equation 4,

$$Loc = \delta_1 \cdot \frac{FC_{DC}}{TC_{DC}} + \delta_2 \cdot \frac{WR}{WR_{Max}} + \delta_3 \cdot \frac{1}{1+HL}\,, \tag{4}$$

where $\sum_{i\in\{1..3\}} \delta_i = 1$, $\delta_i > 0$.

α, β, γ and δ are coefficients specifying the impact of the particular monitoring parameters being used in the above formulas. They need to be tuned for the given environment. The above policies are chosen according to the client profile making request and the type of the request. For instance, if the client has defined in the profile that it needs the data as fast as possible than the R_ML policy is chosen.

6 Test Results

Three types of tests has been conducted:

- Monitoring influence tests - showing PLBS impact on the performance of the monitored HSM systems,
- Response time tests - showing the time of PLBS responds to prediction queries,
- Load balancing tests - showing data access requests distribution among the storage nodes in multi user and multi requests data access paradigm.

Table 3. Test environment nodes

name	location	type	CPU	HSM	tape drives	HSM cache [GB]
smok	Krakow	SN	2×Xeon 3.3GHz	HP FSE	4x LTO	2000
worm	Poznan	SN	2×Xeon 2.8GHz	IBM TSM	3x LTO	400
kmd-pilot3	Wroclaw	SN	2×Xeon 2.8GHz	IBM TSM	none	8
kmd2	Krakow	MN	2×Xeon 2.8GHz	na	na	na

SN - Storage Node, MN - Monitoring Node.

The tests have been conducted in the following environment: 4 nodes described in detail in Table 3 and connected via Pionier network with 1Gb links.

The results are presented in the following subsections.

6.1 Influence Tests

In order for the JIMS to retrieve monitoring data from storage nodes a monitoring agent (JIMS MA) (see Fig.1) needs to be present on these nodes. The goal of these tests is to measure the influence to performance of storage system when the JIMS MA is running on the same node. These tests were performed on the smok SN (see Table 3). This HSM system is in production and the measurements were done during periods of low activity. The smok SN is an HP Proliant DL580 server running File System Extender (FSE) under Linux RHEL5. The main disk storage of the server resides on HP EVA8000 disk array and is attached via 2 FC 2Gb/s links. Repeated patterns of simulated users activities were generated by

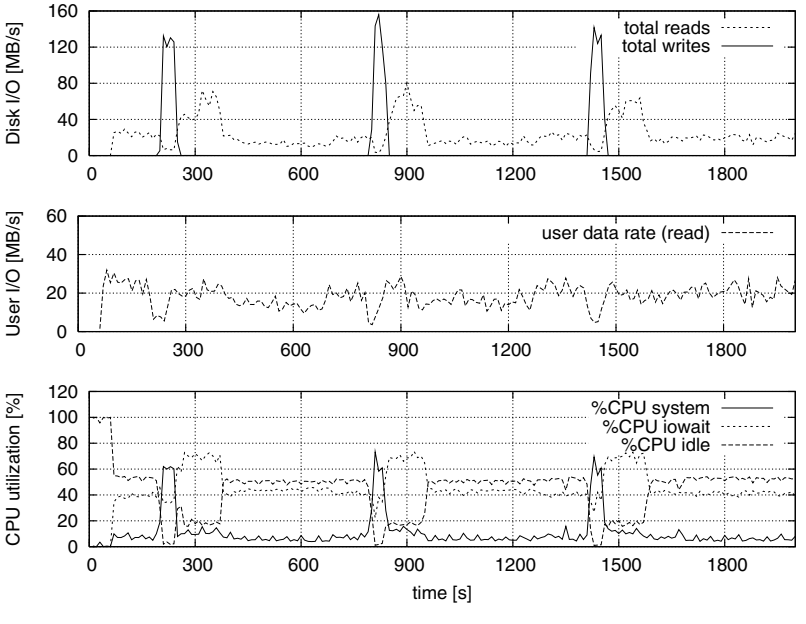


Fig. 3. An example result of monitoring influence test

ftp transfers from other hosts (HSM clients). The JIMS MA performed measurements every 10 minutes. Disk reads and writes generated by the measurements had little impact (maximum 5%) on overall execution times of data transfers to and from clients. An example test result is shown in Fig. 3.

The most influence of JIMS MA activity on users data transfers occurs in short periods when the agent measures disk write performance used to calculate AverageDCWriteRate (see Table 1). The system utilization statistics come from `sar` program. The user data rates were taken from network traffic statistics as there was no other network traffic on the server during the tests.

6.2 Response Tests

Response tests measure the time of processing prediction requests to AMPD. Table 4 presents test results for the implemented replication policies. Each value is taken as an average over 5000 requests. We distinguished two cases: (1) the client is on the same machine that AMPD, (2) the client is located remotely to the AMPD component. For each of these two cases the time of processing a request by AMPD (AMPD columns) and time of processing a request together with communication overhead (client columns) are provided.

Table 4. Time of serving prediction requests

Replication policy	Response time [ms]			
	local		remote	
	AMPD	client	AMPD	client
Reading in shortest time	27.91	31.19	29.46	87.01
Reading from the minimally loaded device	19.42	23.99	21.19	84.60
Writing big files	17.15	21.72	17.41	79.33
Writing to the minimally loaded device	14.50	17.75	16.05	77.60

We can see that the response times are acceptable for all policies and they do not exceed 90 ms for remote clients. The network overhead has great influence on the final response times - without it the processing time is less than 30 ms.

6.3 Load Balancing Test

Load balancing test shows how the requests get distributed among the storage nodes. One monitoring node and three storage nodes have taken part in this test (see Table 3). A script requesting a new replica location prediction and placing data in the result location has been executed on one of the ANs. The script starts new requests until 5 concurrent transfers get present. When a transfer is over another request is started. 5000 requests have been done in 20 hours.

Figure 4 provides results of prediction tests for the W_BF policy with the following coefficient values: $\gamma_1 = 0.6$, $\gamma_2 = 0.2$, $\gamma_3 = 0.1$ and $\gamma_4 = 0.1$. Each point represents the fraction of requests for which a particular host has been selected within the time interval of 1 h.

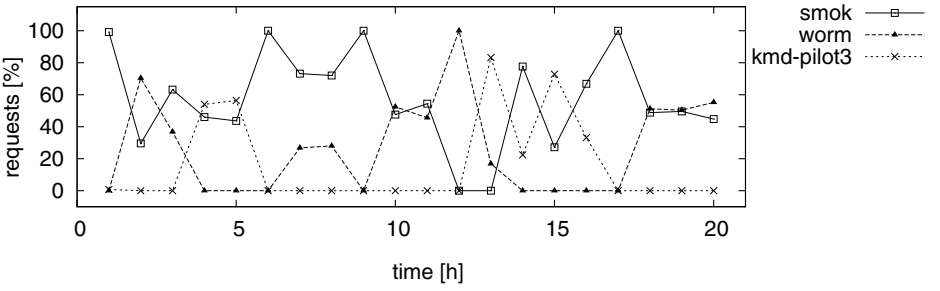


Fig. 4. Replication tests for W_BF policy

We can see that the requests are distributed between the nodes according to their storage processing power - the most powerful host (*smok*) has served the majority of requests. Periodic change of the leader occurs because monitoring data are put into the database in 30 min intervals.

7 Summary and Future Work

In this paper the PLBS subsystem being a part of the NDS system has been presented. The system makes use of replication techniques to increase availability and performance of data access. Monitoring parameters, methods for retrieving them and replication policies have been described. The influence tests showed that the monitoring did not cause essential storage system performance degradation. The system response times are within the tens of milliseconds range which is satisfying. Load balancing test shows that requests get distributed between the nodes proportionally according to their storage processing power. In the near future we plan to extend the set of available policies according to client requirements and to conduct the tests when NDS will be in production.

Acknowledgments

This research is partially supported by the MNiSW grant nr R02055 03 and AGH-UST grant nr 11.11.120.865. Thanks go to Rafał Mikołajczak for valuable help with the TSM system and to NDS partners for sharing storage resources.

References

1. National Data Storage project, Polish MNiSW grant nr R02055 03, https://kmd.pcss.pl
2. Pionier - Polish Optical Internet, http://www.pionier.gov.pl
3. Ranganathan, K., Foster, I.: Identifying Dynamic Replication Strategies for a High-Performance Data Grid. In: Proc. Int. Workshop on Grid Computing, Denver (November 2001)

4. Lamehamedi, H., Szymański, B., Deelman, E.: Data Replication Strategies in Grid Environments, pp. 378–383. IEEE Computer Science Press, Los Alamitos (2002)
5. Park, S., Kim, J., Ko, Y., Yoon, W.: Dynamic Data Grid Replication Strategy Based on Internet Hierarchy. In: Li, M., Sun, X.-H., Deng, Q.-n., Ni, J. (eds.) GCC 2003. LNCS, vol. 3033, pp. 838–846. Springer, Heidelberg (2004)
6. Rahman, R.M., Barker, K., Alhajj, R.: A Predictive Technique for Replica Selection in Grid Environment. In: 7th IEEE Int. Symp. on Cluster Computing and the Grid. IEEE Computer Society, Los Alamitos (2007)
7. Li, J.: A Replica Selection Approach based on Prediction in Data Grid. In: Proc. Third Int. Conf. on Semantics, Knowledge and Grid - SKG 2007, Xi'an, Shan Xi, China, October 29-31, pp. 274–277 (2007)
8. Rahman, R.M., Barker, K., Alhajj, R.: Replica placement Strategies in Data Grid. J. Grid Computing 6, 103–123 (2008)
9. Rasool, Q., Li, J., Oreku, G.S., Zhang, S., Yang, D.: A load balancing replica placement strategy in Data Grid. In: Pichappan, P., Abraham, A. (eds.) Third IEEE Int. Conf. on Digital Information Management (ICDIM), Proc. IEEE 2008, London, UK, November 13-16, pp. 751–756 (2008)
10. Słota, R., Skitał, L., Nikolow, D., Kitowski, J.: Algorithms for Automatic Data Replication in Grid Environment. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 707–714. Springer, Heidelberg (2006)
11. Nikolow, D., Słota, R., Kitowski, J.: Grid Services for HSM Systems Monitoring. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 321–330. Springer, Heidelberg (2008)
12. Zieliński, K., Jarząb, M., Balos, K., Wieczorek, D.: Open Interface for Autonomic Management of Virtualized Resources in Complex Systems - Construction Methodology. FGCS 24(5), 390–401 (2008)

Churn Tolerant Virtual Organization File System for Grids*

Leif Lindbäck[1], Vladimir Vlassov[1],
Shahab Mokarizadeh[1], and Gabriele Violino[2,**]

[1] Royal Institute of Technology (KTH), Stockholm, Sweden
[2] Net Result AB, Stockholm, Sweden

Abstract. A Grid computing environment allows forming Virtual Organizations (VOs) to aggregate and share resources. We present a VO File System (VOFS) which is a VO-aware distributed file system that allows VO members to share files. VOFS supports access and location transparency by maintaining a common file namespace, which is decentralized in order to improve robustness. VOFS includes a P2P system of file servers, a VO membership service and a policy and role based security mechanism. VOFS can be mounted to a local file system in order to access files using POSIX file API. VOFS can operate in a dynamic Grid environment (e.g. desktop Grids) since it tolerates unplanned resource arrival and departure (churn) while maintaining a single uniform namespace. It supports transparent disconnected operations that allow the user to work on files while being disconnected.

Keywords: Grid file system; peer-to-peer; security; namespace.

1 Introduction

A Grid computing environment allows forming Virtual Organizations (VOs). A VO is a collection of users or institutions that pools their resources into a single virtual administrative domain. A VO File System (VOFS) aggregates data objects (files, directories and disk space) exposed by VO members. *Expose* here means to make a data object accessible via VOFS. One major challenge in such a file system is namespace management. Uniform and globally unique path names should be associated with data objects [1]. Uniform here means access and location transparency of exposed data objects, and the same view of the file system at all nodes. This requires mapping a *logical name* of a file in VOFS namespace to its physical location. The global nature of grids enforces logical names to be uniform across different administrative domains. In this work we consider ad-hoc grids and propose a user-level VOFS that allows creating and maintaining work spaces by exposing and sharing data objects by VO members. The proposed VOFS has the following features.

* This research is supported by the FP6 Project Grid4All funded by the European Commission (Contract IST-2006-034567).

** Gabriele Violino was at the Royal Institute of Technology while doing this work.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 194–203, 2010.

1. VOFS includes a security mechanism that protects exposed data objects from unauthorized access, and includes VO membership management, authentication and role-based authorization according to VO policies;
2. VOFS maintains a uniform namespace despite of unplanned resource arrival and departure (churn);
3. VOFS allows ordinary applications to access the VOFS using a standard POSIX file API, i.e. the applications do not need to be modified to access files in the VOFS;
4. VOFS is easy to use for non-experienced users;
5. VOFS can operate under any OS that has WebDAV [2] mount support, e.g. MS Windows, Linux, Mac OS X.
6. VOFS supports transparent disconnected operations that allow the user to work on files while being disconnected.

2 Overview

This work builds on our previous work presented in [3] that proposed three ways of maintaining the namespace: a centralized name service; a distributed directory; and a DHT-based name service. In [3] we have presented VOFS with the centralized name service that has the major disadvantage to induce a single point of failure and a potential performance bottleneck. In this paper, we propose to build VOFS with a namespace maintained as a *distributed directory* where the namespace information is distributed among peers. In this design every peer can potentially learn the entire namespace (i.e. location of exposed data objects) via a gossiping mechanism. The data objects can be exposed to any path in VOFS. An exposed directory offers disk space which is used by VO members to create new objects.

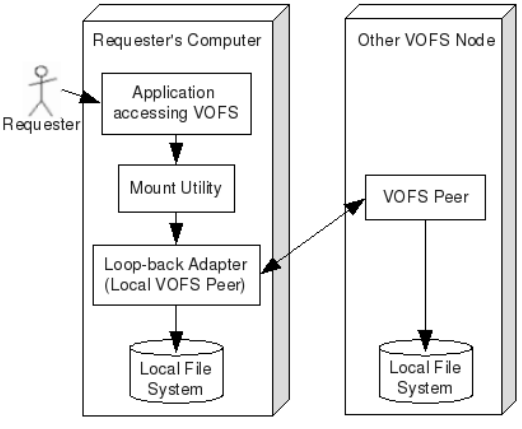


Fig. 1. Schematic view of VOFS architecture

Each peer runs a file server that provides and controls access to data objects exposed from the local node, see Fig. 1. Access to the exposed objects is achieved by mounting the local VOFS peer to a mount point, e.g. a local path. We use the WebDAV protocol [2] to access and transfer files between peers. Use of WebDAV allows accessing VOFS through any mount utility supporting WebDAV, e.g davfs2 [4], which offers a POSIX compliant API. Once mounted, access to VOFS is no different from access to local file system.

3 VOFS Namespace and File Tree

VOFS is formed as an ordinary file tree. Exposed objects are given logical names, which are paths in VOFS. The VOFS namespace is a set of mappings of logical names to physical locations. A VOFS path may include names of *virtual directories*, which are not hosted by any peer, i.e. they do not exist. Thus, VOFS consists of exposed real data objects and virtual directories that may contain virtual directories and real data objects.

Initially, the VOFS tree contains only the root, which is initially virtual. The VOFS namespace, hence the VOFS tree, is formed explicitly and gradually as a result of exposing and unexposing data objects. Virtual directories help to maintain the namespace. If to assume that all directories in the VOFS tree are real (i.e. physically exist), then unexposing a real directory may cause partitioning of the tree as the data objects under the unexposed directory can not be properly identified. This motivates introducing virtual directories. The unexposed real directory becomes virtual; and names of all objects under it remain unchanged.

When looking up location of an object given its fully-specified VOFS path, a longest prefix match is done. The object can be accessed if the exposing peer is online despite of whether other peers are online or not.

Mappings of logical names to physical locations are the major metadata of VOFS. The metadata associates exported data objects with paths in the VOFS namespace. The same metadata are kept at every node in two tables: remote.db, which stores location information of data objects exposed by other peers; and local.db that stores location information of objects exposed by this peer. When a data object is exposed, the exposing peer adds a pair of local file system path and VOFS path to the local.db table while all other peers adds a pair of VOFS path and physical host address to their remote.db table. When a data object is unexposed, this information is removed from all peers. The namespace changes only when peers perform expose or unexpose operations. Peers communicate metadata by gossiping as explained below. All peers know the entire namespace, i.e which data objects are exposed and who exposes them.

3.1 Algorithm for Namespace Updates

To transfer namespace updates between peers we use a gossip algorithm based on the *lazy probabilistic broadcast algorithm* described in [5]. When a peer updates the namespace it sends an *update* message to all or some of its neighbours. Each peer that receives an *update* message forwards it to all or some of its neighbours. There will be no loops since a peer never sends the same message twice.

The following recovery mechanism is used when messages are lost. Original sender id and a sequence number are attached to each message. Since there is FIFO delivery of messages, if a peer receives a message with a sequence number larger than the previous number plus one, it knows that some messages were lost. It will then send a *require* message to a subset of its neighbours, indicating which message was lost and which peer is requiring it. A peer, which receives the *require* message checks if it has the required message. If yes, it sends the required

update message to the requiring peer. If not, it forwards the *require* message to its neighbours. *Require* messages are forwarded only a specified number of times. Each peer maintains information about transmitted messages on its hard disk.

The gossip algorithm described above is used only for namespace updates. All other communication, e.g. file transfer, involve only two peers. Due to gossiping, there is no need to search for data objects since each peer maintains its own view of the namespace which is almost the same as views of other peers, even though there might be some inconsistencies between views caused by update latency.

4 VOFS Peers

A user exposing data objects must run a VOFS peer on her computer; while a user accessing VOFS does not need to run a peer. In the latter case, the user must know an address of a peer to mount it and to access VOFS. If the user runs a peer, it can be mounted to become the entry point to VOFS. In this case, there is no need to keep addresses of well-known mount points. All VOFS peers provide (un)expose, join, mount, and cache services, which are described below. The services can be accessed through the GUI of the VOFS peer.

(Un)Expose. When exposing, the user defines a data object to be exposed and specifies its VOFS path. The expose service stores the logical-to-physical name mapping in the local table and initiates the update gossip algorithm. If the specified path does not exist, virtual directories are introduced to allow traversing the tree from root to the exposed data object. The root of VOFS is always `/`. It can be either virtual or mapped to a real directory. Name collision occurs when the user tries to assign a name which is already taken. Name collision is resolved as follows: if the data object to be exposed is a file, its mapping overrides the mapping of the object previously exposed with the same name; in case of directories exposed with the same name, their contents are merged.

Join. When a user starts a VOFS peer, the peer joins the P2P VOFS system. At startup, the peer downloads a list of all VO peers from the VO Membership Service (VOMS) described in 5.1. Then the peer connects to some other peers selected from the list. The chosen peers and the new peer become neighbours. In the current VOFS prototype, selection of neighbours is random, but it could be done in a sophisticated way. They also exchange their VOFS views stored in their local and remote metadata tables described earlier. It is possible for the user to manually edit a peer's neighbour list through the GUI of the VOFS peer.

Mount. The user can mount VOFS with any mount utility supporting WebDAV; therefore we have not developed any mount utility; instead, we use davfs [4] on Linux and NetDrive [6] on MS Windows. VOFS has not been tested on other OSs but Mac OS X has WebDAV support built in.

Once the VOFS is mounted, all POSIX file API is supported for manipulating data objects (provided the mount utility offers a POSIX API). The mount utility will translate the POSIX calls to WebDAV calls to the VOFS peer.

Cache. Each VOFS peer maintains a file cache. Read and write latency over network is compensated by the caching mechanism, which also allows offline work. VOFS uses *last write wins* reconciliation policy (a traditional file system policy for concurrent writes), which can be replaced by a more sophisticated reconciliation policy implemented using, for example, Telex [7]. The cached copy is checked for update (compared to the master copy) when the file is read. When a file is written the new content is stored in the cache and sent to the exposing peer, which informs all other peers caching the file about the update. Also directory listings are cached, but unlike files they have an expiry time.

5 Security

The security infrastructure is based on the XACML authorization model [8]. Its goal is to provide authentication and authorization. When authenticating, the user's credentials are checked and the user gets a token, which can be used to prove her identity in authorization checks. Authorization grants that users can only access resources to which they have right according to VO policies. Authorization is policy-based, policies are expressed in XACML.

5.1 Components

The VOFS security infrastructure is built of the following components.

Virtual Organization Membership Service, VOMS keeps a database of users and roles in the VO. It has a web based management interface for updating this data. This interface is protected by a PEP. The VOMS is also responsible for authenticating users.

Policy Enforcement Point, PEP protects a resource (VOFS peer, VOMS, PAP). Each resource has a local PEP. The PEP sends authorization requests to the PDP and caches the answers. To improve performance the PDP answers not only to the request sent by the PEP, but to requests with the same subject and resource with all existing actions.

Policy Decision Point, PDP evaluates requests from PEPs according to the policies in PR. Policies are cached in memory. Invalidation of the PDP's cache also invalidates all PEP's caches.

Policy Information Point, PIP contacts VOMS to validate the user's identy and get the user's roles. The answer from VOMS is cached.

Policy Administration Point, PAP is a server that makes updates to PR. The PAP is protected by a PEP. A client to the PAP is provided.

Policy Repository, PR stores policies as XACML files.

We suppose that except for PEP there will be one instance of each component per VO. Each PEP should be placed on the same host as the protected resource.

5.2 Scenarios

A typical scenario of interactions between security components and VOFS peers is depicted in Fig. 2. We distinguish four different phases: creating users and roles, setting security policies, authentication and access control.

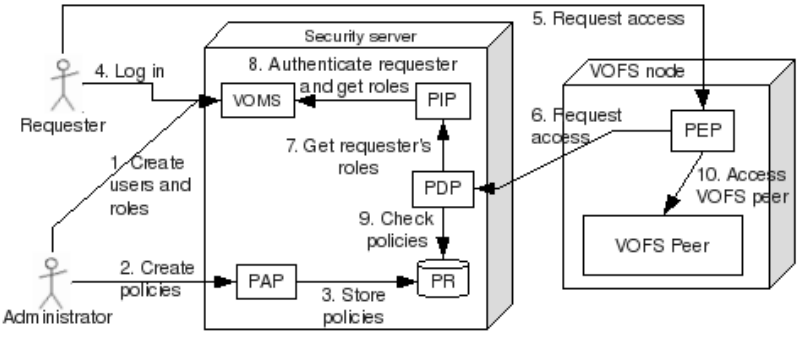


Fig. 2. Security components

Creating users and roles (1) VO admin uses VOMS to create users and roles.

Setting policies (2) The administrator uses PAP to create policies. (3) PAP stores the policies in PR. PAP will invalidate PDP's cache. It can be specified in a policy when it is valid. This can be specified as time, date and day of week ranges and any combination of these.

Authentication (4) The requester logs in to VOMS, using a web based interface. If the requester is authenticated, VOMS returns a token that is stored on the requester's computer.

VOFS access (5) The requester uses an application that accesses VOFS. The mount utility sends the token along with the call to VOFS. The call is intercepted by the PEP protecting the VOFS peer. (6) PEP asks PDP whether the requester is allowed to access the peer. (7) PDP asks PIP for the requester's roles. (8) PIP contacts VOMS to check if the token is valid and to get the requester's roles. (9) PDP evaluates the policies stored in PR. (10) If access was granted, the call is let through to the VOFS peer.

5.3 Secure Communication

The goals of secure communication are

1. To guarantee that PEPs get answers from the correct PDP;
2. To guarantee that PDP gets answer from the correct VOMS;
3. To guarantee that the token identifying a user is not stolen. If it is stolen it can be used to impersonate that user.

The first two goals can be met using certificates to identify PDP and VOMS. Regarding the third goal, there are the following risks that the token is stolen:

1. During transfer (this risk is eliminated with encrypted communication);
2. From the user's computer;
3. By a malicious node pretending to be a VOFS peer;

The second risk can be reduced if the VOMS encrypts the token with the user's public key. The third risk is eliminated if peers only communicate with other

peers that can prove they are part of VOFS. This can be achieved by using a certificate signed by a trusted certificate authority, CA. Each peer gets its certificate from the VOMS at startup. None of the above solutions require the user to be aware that certificates are used. This makes the VOFS easy to use also for non-experienced users.

6 VOFS Prototype

The prototype is implemented using Java Servlets, Fig. 3 shows its main components. It comes bundled with Apache Tomcat.

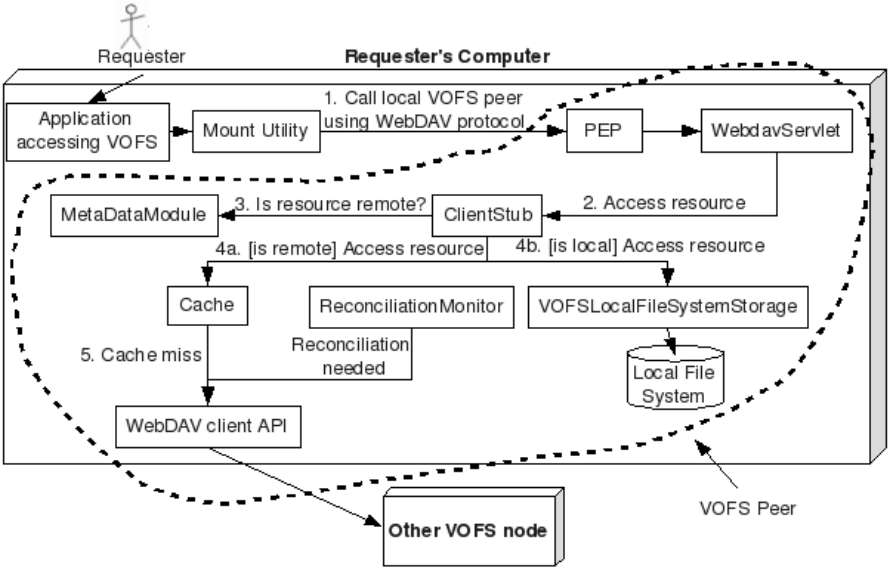


Fig. 3. VOFS implementation

PEP is a servlet filter that intercepts all incoming requests. It translates the WebDAV method of the call to a VOFS operation and calls PDP to check if the operation is allowed. If not, an HTTP 403 (forbidden) code is returned.
WebdavServlet The access point for remote peers and the local mount utility.
ClientStub Forwards requests to the correct component.
MetaDataModule Keeps metadata and performs longest prefix match.
LocalFileSystemStorage Provides access to exposed files and directories.
Cache Caches remote data objects. If a searched object is not in the cache the call is forwarded to the remote peer hosting it. The returned object is cached.
WebDAV client api Reads and writes data objects from remote peers.
ReconciliationMonitor Continuously monitors cache to see if an item in cache is newer than the master, if so updates the master.

7 Related Work

Sprite Network File System [9] is a distributed file system similar in some aspects to VOFS. Meta-data in VOFS with decentralized name service is handled in a similar way to Sprite. A main difference between VOFS and Sprite is that Sprite does not handle partitioning of the file tree since lookup starts from root and proceeds downwards; whereas in VOFS longest prefix match is done on the entire path. Support for virtual directories and disconnected operation makes VOFS churn tolerant.

There exist P2P file systems, e.g. OceanStore [10], which were developed as a storage for file sharing. A typical P2P file system has no support for neither POSIX file API, nor security. Grid file systems in contrast to P2P file sharing systems strongly require authentication and authorization to protect files from unauthorised access. VOFS allows the VO members to define and set VO security policies to be enforced by the VOFS security infrastructure.

Examples of Grid file systems include gLite file catalogs [11], Gfarm [12], and Distributed File Services, DFS [13]. The gLite *file catalogue service* [11] is used to maintain location information about files and their replicas. In contrast to VOFS, gLite catalogue service is centralized and is a single point of failure. The Gfarm file system [12] uses a virtual tree and virtual directories mapped to physical files by a metadata server, like the *centralized* solution described in [3]. Gfarm is designed to be very scalable; however, its metadata server can become a bottleneck and is a single point of failure since it is not replicated in contrast to VOFS. DFS [13] is a P2P file and storage system that can be integrated with a Grid security mechanism. DFS, in contrast to the presented VOFS, has no hierarchical namespace, but instead offers two P2P networks: one for storage space and one for names and metadata. DFS is implemented using FUSE [14] that limits its usage only to Linux, while the WebDAV-based VOFS can run on multiple (if not all) operating system, e.g. MS Windows, Linux, Mac OS X.

8 Performance Evaluation

We have evaluated performance of namespace updates, file transfer and file lookup. The nodes are PCs with 1.86 GHz Intel Centrino CPUs and 1 GB RAM on a dedicated 100 Mbps LAN.

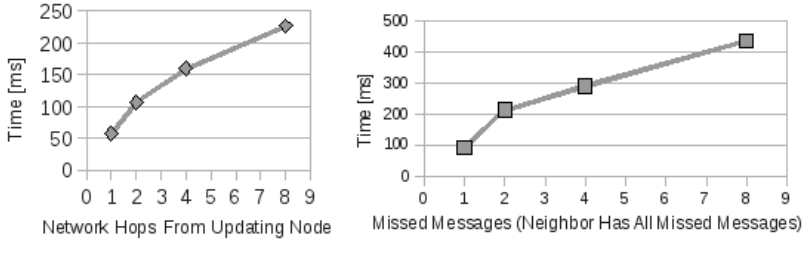


Fig. 4. Namespace update performance

We have done two measurements of the namespace update algorithm (see Fig. 4). The first measurement concerns updates without lost messages. It shows how long time it takes for an update message to reach a node that is one, two, four and eight network hops away from the updating node. The second measurement shows recovery of missed namespace update messages due to a node being disconnected from the node performing the updates. Figure 4 shows how long it takes to get information about all namespace updates performed while the node was disconnected. This is measured whith one, two, four and eight missed update messages. The time is reduced if all lost messages are required and resent with one message, now there is one require and one resend per lost message. Figure 4 shows that the algorithm scales well.

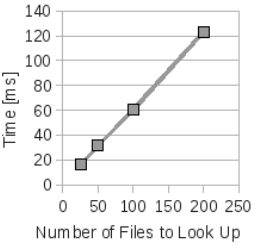


Fig. 5. Lookup performance

Figure 5 shows how long time it takes to find out which node exposes a given file. This is a local operation, since all nodes have information about the entire namespace. The lookup time is about 0.7 ms per file no matter how many files are looked up.

The read test copies 100 files from a remote peer to the local file system (outside VOFS). The file cache is big enough to contain all files. The results are presented in Fig. 6. The figure also depicts timings for copying files within the local file system in order to compare performances of the local file system and VOFS. Bandwidth when transferring smaller files is lower because overhead takes proportionally more time. The overhead is mainly due to that the mount utility (davfs2) reads file properties before transferring files.

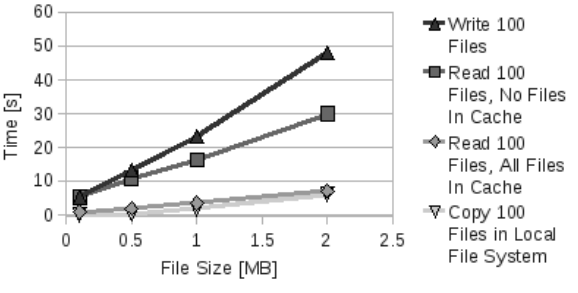


Fig. 6. File transfer performance

The write test copies 100 files from the local file system (outside VOFS) to a remote peer. Results of write test are in Fig. 6. Cache does not speed up performance since file content is written both to cache and to remote peer.

9 Conclusion

We have presented a churn tolerant VOFS that maintains a uniform namespace in a dynamic environment, that is when nodes frequently join or leave the VOFS. The VOFS provides a shared workspace for VO members and it is easy to use. It includes VO membership management, authentication and authorization. The VOFS Prototype is available at `http://www.isk.kth.se/~leifl/vofs/`

Acknowledgement

Special thanks to Chen Xing (chenxing@kth.se) who implemented the first version of the VOMS and the gossip based protocol.

References

1. Anderson, O.T., et al.: Global namespace for files. IBM systems Journal 43(4), 702–722 (2004)
2. WebDAV Community, http://www.webdav.org/
3. Mizani, H.R., Zheng, L., Vlassov, V., Popov, K.: Design and Implementation of Virtual Organization File System for Dynamic VOs. In: Proc. of the 11th IEEE Int. Conf. on Computational Science and Engineering, Workshops, pp. 77–82 (2008)
4. davfs2, mount utility for WebDAV on Linux, http://dav.sourceforge.net/
5. Guerraoui, R., Rodrigues, L.: Introduction to Reliable Distributed Programming. Springer, Heidelberg (2006)
6. NetDrive, mount utility for WebDAV and FTP, http://www.netdrive.net/
7. Benmouffok, L., Busca, J., Marqus, J.M., Shapiro, M., Sutra, P., Tsoukalas, G.: Telex: Principled System Support for Write-Sharing in Collaborative Applications. Research Report, INRIA RR-6546 (2008)
8. Lang, B., Foster, I., Siebenlist, F., Ananthakrishnan, R., Freeman, T.: A Multi-policy Authorization Framework for Grid Security. In: Proc. of the Fifth IEEE Symposium on Network Computing and Application, pp. 269–272 (2006)
9. Welch, B., Ousterhout, J.: Prefix Tables: A Simple Mechanism for Locating Files in a Distributed System. Report No. UCB/CSD 56/261, Computer Science Division, University of California, Berkeley, California (1985)
10. Rhea, S., Eaton, P., Geels, D., Weatherspoon, H., Zhao, B., Kubiatowicz, J.: Pond: the OceanStore Prototype. In: Proc. of the 2nd USENIX Conf. on File and Storage Technologies, pp. 1–14 (2003)
11. gLite, middleware for grid computing, http://glite.web.cern.ch/glite/
12. Tatebe, O., Sekiguchi, S., Morita, Y., Soda, N., Matsuoka, S.: GFARM V2: A Grid File System that Supports High-Performance Distributed and Parallel Data Computing. In: Computing in High Energy Physics and Nuclear Physics, Interlaken, Switzerland, p. 1172 (2004)
13. Chazapis, A., Tsoukalas, G., Verigakis, G., Kourtis, K., Sotiropoulos, A., Koziris, N.: Global-scale peer-to-peer file services with DFS. In: 2007 8th IEEE/ACM Int. Conf. on Grid Computing, pp. 251–258 (2007)
14. FUSE: Filesystem in Userspace, http://fuse.sourceforge.net/

Quasi-random Approach in the Grid Application SALUTE*

Emanouil Atanassov, Aneta Karaivanova, and Todor Gurov

Institute for Parallel Processing - Bulgarian Academy of Sciences,
Acad. G. Bonchev St., Bl.25A, 1113 Sofia, Bulgaria
{emanouil,anet,gurov}@parallel.bas.bg

Abstract. **S**tochastic **A**Lgorithms for **U**ltra-fast **T**ransport in s**E**miconductors **(SALUTE)** is a Grid application which integrates a set of novel Monte Carlo, quasi-Monte Carlo and hybrid algorithms for solving various computationally intensive problems important for industry (design of modern semiconductor devices). SALUTE studies memory and quantum effects during the femtosecond relaxation process due to electron-phonon interaction in one-band semiconductors or quantum wires.

There are two main reasons for running this application on the Grid: (i) quantum problems are very computationally intensive; (ii) the inherently parallel nature of Monte Carlo applications makes efficient use of Grid resources.

In this paper we study the quasirandom approach in SALUTE, using the scrambled Halton, Sobol and Niederreiter sequences. A large number of tests have been performed on the SEEGRID grid infrastructure using specially developed grid implementation scheme. Novel results for energy and density distribution, obtained in the inhomogeneous case with applied electric field are presented.

Keywords: Grid computing, Monte Carlo methods, Quasi-Monte Carlo, Scrambled Halton, Sobol and Niederreiter sequences, Ultra-fast carrier transport.

1 Introduction

The Monte Carlo Methods (MCMs) for quantum transport in semiconductors and semiconductor devices have been actively developed during the last two decades, [3,9,12]. These Monte Carlo calculations need large amount of computational power and the reason is as follows: If temporal or spatial scales become short, the evolution of the semiconductor carriers cannot be described in terms of the Boltzmann transport and therefore a quantum description is needed. Let us note that in contrast to the semiclassical transport when the kernel is positive, the kernel in quantum transport can have negative values. The arising problem, sometimes referred to as the "negative sign problem", leads to additional computational efforts for obtaining the desired solution.

* Supported by the Ministry of Education and Science of Bulgaria under Grant No. DO02-146/2008.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 204–213, 2010.

Usually Monte Carlo methods have convergence rate of $O(N^{-1/2})$ One generic approach to improving the convergence of MCMs has been the use of highly uniform random numbers (called quasirandom sequences - QRNs) in place of the usual pseudo random numbers (PRNs). The methods based on quasirandom sequences are called quasi-Monte Carlo methods (QMCMs), a term proposed by H. Niederreiter. The successful application of the QMCMs for multiple integrals in financial mathematics (late 90s) was followed by an active work in the area of QMCMs for Markov chain based problems.

In this paper we present quasirandom approach for ultrafast carrier transport simulation. Due to correlation, the direct quasirandom variants of Monte Carlo methods do not give adequate results. Instead of them, we propose hybrid algorithms with pseudorandom numbers and scrambled quasirandom sequences. We use scrambled modified Halton [2], scrambled Sobol [1] and scrambled Niederreiter sequences. In this paper we present also our grid implementation scheme which uses not only the computational capacity of the grid but also the available grid services in a very efficient way. With this scheme we were able to obtain new estimates about important physical quantities. The paper is organised as follows: Section 2 describes very briefly the problem and the Monte Carlo algorithms, section 3 presents the quasirandom sequences and hybrid algorithms, section 4 describes the grid implementation scheme, section 4 contains the new estimates and performance analisys.

2 Background (Brief Description of SALUTE)

The first version of SALUTE was designed in 2005 as a set of Monte Carlo algorithms for simulation of ultra-fast carrier transport in semiconductors together with simple Grid implementation, [3,4]. Later on, we extended the the area of application (quantum wires), the algorithms (more complicated equations to be solved) and the implementation scheme. In this paper we present the quasirandom approach in SALUTE and discuss the new results. The physical model describes a femtosecond relaxation process of optically excited electrons which interact with phonons in one-band semiconductor, [13]. The interaction with phonons is switched on after a laser pulse creates an initial electron distribution. In our model we consider a low-density regime, where the interaction with phonons dominates the carrier-carrier interaction. Two cases are studied using SALUTE: electron evolution in presence and in absence of electric field.

As a mathematical model we consider Wigner equation for the nanometer and femtosecond transport regime. In the homogeneous case we solve a version of the Wigner equation called Levinson (with finite lifetime evolution), or Barker-Ferry equation (with infinite lifetime evolution). Another formulation of the Wigner equation considers inhomogeneous case when the electron evolution depends on the energy and space coordinates. Particularly we consider a quantum wire, where the carriers are confined in the plane normal to the wire by infinite potentials. The initial condition is assumed both in energy and space coordinates. We recall the integral form of the quantum-kinetic equation, [13]:

$$f_w(z,k_z,t) = f_w(z - \frac{\hbar k_z}{m}t + \frac{\hbar \mathbf{F}}{2m}t^2, k_z, 0) + \int_0^t \partial t'' \int_{t''}^t \partial t' \int d\mathbf{q}'_\perp \int dk'_z \times \quad (1)$$

$$\left[S(k'_z,k_z,t',t'',\mathbf{q}'_\perp) f_w \left(z - \frac{\hbar k_z}{m}(t-t'') + \frac{\hbar \mathbf{F}}{2m}(t^2 - t''^2) + \frac{\hbar q'_z}{2m}(t'-t''), k'_z, t'' \right) \right.$$

$$\left. - S(k_z,k'_z,t',t''\mathbf{q}'_\perp) f_w \left(z - \frac{\hbar k_z}{m}(t-t'') + \frac{\hbar \mathbf{F}}{2m}(t^2 - t''^2) - \frac{\hbar q'_z}{2m}(t'-t''), k_z, t'' \right) \right]$$

$$S(k'_z,k_z,t',t'',\mathbf{q}'_\perp) = \frac{2V}{(2\pi)^3} |G(\mathbf{q}'_\perp)\mathcal{F}(\mathbf{q}'_\perp, k_z - k'_z)|^2 \times$$

$$\left[(n(\mathbf{q}')+1) cos \left(\frac{\epsilon(k_z) - \epsilon(k'_z) + \hbar\omega_{\mathbf{q}'}}{\hbar}(t'-t'') + \frac{\hbar}{2m}\mathbf{F}.q'_z(t'^2 - t''^2) \right) \right.$$

$$\left. + n(\mathbf{q}') cos \left(\frac{\epsilon(k_z) - \epsilon(k'_z) - \hbar\omega_{\mathbf{q}'}}{\hbar}(t'-t'') + \frac{\hbar}{2m}\mathbf{F}.q'_z(t'^2 - t''^2) \right) \right]$$

Here, $f_w(z,k_z,t)$ is the Wigner function described in the $2D$ phase space of the carrier wave vector k_z and the position z, and t is the evolution time.

$\mathbf{F} = e\mathbf{E}/\hbar$, where $\mathbf{E}$ is a homogeneous electric field along the direction of the wire z, e being the electron charge and $\hbar$ - the Plank's constant.

$n_{\mathbf{q}'} = 1/(\exp(\hbar\omega_{\mathbf{q}'}/\mathcal{K}T) - 1)$ is the Bose function, where $\mathcal{K}$ is the Boltzmann constant and T is the temperature of the crystal, corresponds to an equilibrium distributed phonon bath.

$\hbar\omega_{\mathbf{q}'}$ is the phonon energy which generally depends on $\mathbf{q}' = \mathbf{q}'_\perp + q'_z = \mathbf{q}'_\perp + (k_z - k'_z)$, and $\varepsilon(k_z) = (\hbar^2 k_z^2)/2m$ is the electron energy.

$\mathcal{F}$ is obtained from the Fröhlich electron-phonon coupling by recalling the factor $i\hbar$ in the interaction Hamiltonian:

$$\mathcal{F}(\mathbf{q}'_\perp, k_z - k'_z) = -\left[\frac{2\pi e^2 \omega_{\mathbf{q}'}}{\hbar V} \left(\frac{1}{\varepsilon_\infty} - \frac{1}{\varepsilon_s} \right) \frac{1}{(\mathbf{q}')^2} \right]^{\frac{1}{2}},$$

where (ε_∞) and (ε_s) are the optical and static dielectric constants. The shape of the wire affects the electron-phonon coupling through the factor

$$G(\mathbf{q}'_\perp) = \int d\mathbf{r}_\perp e^{i\mathbf{q}'_\perp \mathbf{r}_\perp} |\Psi(\mathbf{r}_\perp)|^2,$$

where Ψ is the ground state of the electron system in the plane normal to the wire.

Detailed description of MCMs for this problem can be found in [3,11,12]. Let us mention that MCMs have the advantage that MCMs estimate directly the necessary quantities, i.e. without calculating the solution of the Wigner function in the whole domain. The serious problem with MCMs is the large variance of the random variable which is proportional to the $exp(T^2)$ where T is the evolution time. As the physicists are interested in the quatum effects for large evolution time, the problem becomes computationally very intensive - we have to perform bilions of trajectories in order to obtain reasonable results. This was our motivation for applying quasirandom approach and using computational grid.

3 Quasirandom Approach in SALUTE

Quasi-Monte Carlo methods proved to be efficient in many areas ranging from physics to economy. We have applied quasirandom approach for studying quantum effects during ultrafast carrier transport in semiconductors and quantum wires in order to reduce the error and to speedup the computations. Next, we have used scrambled sequences for two main reasons: (i) the problem is very complicated (the use of scrambling corrects the correlation problem found when we have used a purely quasi-Monte Carlo algorithm), and, (ii) the Grid implementation needs parallel streams.

The computational Grid proved to be very efficient computing model. The Grid goes well beyond simple communication between computers and aims ultimately to turn the global network of computers into one vast computational resource, [10]. Using the Grid is especially useful for Monte Carlo applications as there the amount of similar calculations that has to be done is huge. Technically Grid coordinates resources which are not a subject to central administrative control and utilizes general-purpose protocols. Another distinction is that a Grid could in principle have access to parallel computers, clusters, farms, local Grids, even Internet computing solutions, and would choose the appropriate tool for a given calculation. In this sense, the Grid is the most generalised form of distributed computing. One major advantage of Monte Carlo methods is that they are usually very easy to be parallelized. This is, in principal, also true of quasi-Monte Carlo methods. However, the successful parallel implementation of a quasi-Monte Carlo application depends crucially on various quality aspects of the parallel quasirandom sequences used [7,8]. Much of the recent work on parallelizing quasi- Monte Carlo methods has been aimed at splitting a quasirandom sequence into many subsequences which are then used independently on the various parallel processes, for example in [1,2,5]. This method works well for the parallelization of pseudorandom numbers, but due to the nature of quality in quasirandom numbers, this technique has some difficulties. Our algorithms are based on scrambling (suitable for heterogeneous computing environments).

3.1 Quasirandom Sequences

The main reason to use low-discrepancy sequences instead of pseudorandom numbers, is that one can achieve convergence rates that are close to $O(N^{-1})$, where N is the number of trajectories. The convergence rate of the integration depends on measures of non-uniformity of the sequence. The most popular measure is the star discrepancy, due to the famous Koksma-Hlawka inequality, [6]. Sequences that have the best possible order $O(log(N)^s/N)$ are called *low-discrepancy sequences* or *quasirandom sequences*. The most popular sequences for practical applications are those of Halton, Sobol, Niederraiter and Faure, not necessarily in this order. In the computations presented in this paper, we use scrambled Halton, Sobol and Niederreiter sequences.

Halton sequence. We use the modified Halton sequences introduced in [2] for which the discrepancy has a very small leading term. This construction is

based on the existence of some numebrs, called "admissible". Here we recall the definitions of admissible numbers and modified Halton sequence.

Definition 1. *Let $p_1, \ldots, p_s$ be distinct primes. The integers $k_1, \ldots, k_s$ are called admissible for them, if $p_i \nmid ki$ and for each set of integers $m_1, \ldots, m_s$, $p_i \nmid m_i$, there exists a set of integers $\alpha_1, \ldots, \alpha_s$, satisfying the congruences*

$$k_i{}^{\alpha_i} \prod_{1 \le j \le n,\ j \ne i} p_j{}^{\alpha_j} \equiv m_i (mod\ p_i), \quad i = 1, \ldots, s.$$

Definition 2. *Let $p_1, \ldots, p_s$ be distinct primes, and the integers $k_1, ...k_s$ are admissible for them. The modified Halton sequence $\sigma(p_1, \ldots, p_s; k_1, \ldots, k_s) = \{(x_n{}^{(1)}, \ldots, x_n{}^{(s)})\} n = \{0, \infty\}$ is constructed by setting each sequence $\{x_n{}^{(i)}\}_{n=0}^{\infty}$ to be a generalized Van der Corput - Halton sequence in base p_i, with the sequence of permutations $\tau_j{}^{(i)}(t)$ to be the reminder of $tk_i{}^j$ modulo p_i, $\tau_j{}^{(i)}(t) \in \{0, \ldots, p_i - 1\}$.*

Determining "admissible" generation of modified Halton sequence can be found in [2]. In the experiments described in this paper we use the following scrambling: We change the formulas for the permutations as

$$\tau_j{}^{(i)}(t) \equiv tk_i{}^{(j+1)} + b_j{}^{(i)} (mod\ p_i),$$

where the integers $b_j{}^{(i)}$ are chosen independently in the interval $[0, p_i - 1]$. The scrambled sequence has the same estimate for its discrepancy as if for any integers $m_1, \ldots, m_s$ the congruences

$$k_i{}^{\alpha_i} \prod_{1 \le j \le n,\ j \ne i} p_j{}^{\alpha_j} \equiv m_i (mod\ p_i), \quad i = 1, \ldots, s \tag{2}$$

have a solution, then the same is true for the congruences

$$k_i{}^{\alpha_i + 1} \prod_{1 \le j \le n,\ j \ne i} p_j{}^{\alpha_j} + b_j \equiv m_i (mod\ p_i), \quad i = 1, \ldots, s. \tag{3}$$

The chosen algorithm is very fast, requires a small ammount of memory and generates the terms of sequences with maximal error less that 10^{-14} when 10^6 terms are generated. It shows superior results compared to other Halton generators.

Niederreiter and Sobol sequences. We use the Definition 3 (below), which covers most digital (t, m, s)-nets in base 2. The Sobol sequence is a (t, s)-sequence in base 2 and is a particular case of this definition.

Definition 3. *Let $A_1, \ldots, A_s$ be infinite matrices $A_k = \{a_{ij}^{(k)}\}$, $i, j = 0, 1, \ldots$, with $a_{ij}^{(k)} \in \{0, 1\}$, such that $a_{ii}^{(k)} = 1$ for all i and k, $a_{ij}^{(k)} = 0$ if $i < j$. The $\tau^{(1)}, \ldots, \tau^{(s)}$ are sequences of permutations of the set $\{0, 1\}$. Each non-negative integer n may be represented in the binary number system as*

$$n = \sum_{j-0}^{r} b_i 2^i.$$

Then the nth term of the low-discrepancy sequence σ is defined by

$$x_n^{(k)} = \sum_{j=0}^{r} 2^{-j-1} \tau_j^{(k)} (\oplus_{i=0}^{j} b_i a_{ij}^{(k)}),$$

where by "$\oplus$" we denote the operation "bit-wise addition modulo 2".

To obtain the results presented in this paper we have used the scrambled Sobol and Niederreiter sequences. The algorithm for generating scrambled Sobol sequence is described in [1]. We have modified this algorithm for generating scrambled Niederreiter sequence. We have to note that not all optimizations for Sobol sequence can be applied for Niederreiter (due to the structure of metrices $A's$. This algorithm allows consecutive terms of the scrambled sequence to be obtained with essentially only two operations per coordinate: one floating point addition and one bit-wise xor operation (this omits operations that are needed only once per tuple). This scrambling is achieved at no additional computational cost over that of unscrambled generation as it is accomplished totally in the initialization. In addition, the terms of the sequence are obtained in their normal order, without the usual permutation introduced by Gray code ordering used to minimize the cost of computing the next Sobol element. This algorithm is relatively simple and very suitable for parallel and grid implementation.

3.2 Hybrid Algorithms in SALUTE

We have constructed hybrid Monte Carlo algorithm that uses pseudorandom numbers for some dimensions and scrambled quasirandom numbers for other dimensions. A schematic description of the algorithm is given below, assuming that we only need to compute the Wigner function at one point $(k1, z1)$. In the algorithm, ϵ_1 is the truncation parameter.

- Input: number of trajectories to be used N, relaxation time T, other parameters, describing the initial condition.
- For i from 1 to N sample a trajectory as follows:
  - set time $t := T$, weight $W := 1, k = k_1, z := z_1$
  - prepare the next point of the quasirandom sequence to be used (Niderreiter, Halton or Sobol) $(x_1, x_2, \ldots, x_n)$, with n sufficiently big ($n = 100$ in our case), and set $j = 1$
  - repeat until $t > \epsilon_1$:
    - k is simulated using pseudorandom numbers
    - t', t are simulated using consecutive dimensions of the quasirandom sequence, i.e. the points x_{2j-1}, x_{2j}, by the formulae

$$t_2 := t x_{2j-1}, t_1 := t_2 + x_{2j}(t - t_2), t' := t_1, t = t_2$$

    - multiply the weight: $W := W \star t(t - t_2)$
    - compute the two kernels K_1 and K_2
    - select which one to use with probability proportional to their absolute values.

- multiply the weight: $W := W \star (|K1| + |K2|)sgn(K_m)$ if K_m is the kernel selected
- sample q using a spline approximation of the inverse function
- multiply the weight by the appropriate integral: $W := W \star I$
- modify k, depending on the kernel and the electric field applied: $k_{new} = k - c_3 \star (t - t_2)$ if K_1 was chosen or $k_{new} = k - c_3 \star (t - t_2)$ if K_2 was chosen
- modify $z : z_{new} = z - c_1 \star k \star (t - t_2) - c_2 \star (t - t_2) \star (t + t_2)$
- compute the contribution of this iteration to the Wigner function: add $W \star \psi(z, k)$ to the estimator, where $\psi(z, k)$ is the value of the initial condition
- increment $j := j + 1$

The constructive dimensionality of the algorithm is $4n$, where n is the maximal length of the trajectory. We use $2n$ pseudorandom numbers for each trajectory, and the dimensionality of the Halton sequence is $2n$.

4 Grid Implementation

The computations are performed on the SEE-GRID infrastructure (www.see-grid-sci.eu) which integrates computational and storage resources in South Eastern Europe. Currently there are more than 30 clusters with a total of more than 2000 CPUs and more than 10 TB of storage. The SEE-GRID infrastructure was built using the gLite middleware. Each of the SEE-GRID clusters has the mandatory Grid services: *Computing Element, Worker Nodes, Storage Element* and *MON box*. The Worker Nodes provide the computational resource of the site, and the Storage Element provides the storage resources. The set of services, that are not tied to the specific site are called *core services*. They include VOMS (Virtual organisation management system), MyProxy, R-GMA registry/schema server (distributed data-base), BDII (provides comprehensive information about the resources), WMS (distributes and manages the jobs among the different grid sites), FTS (file transfer service), AMGA (metadata catalog).

4.1 Grid Implementation Scheme

In our grid implementation scheme we incorporated the use of the FTS and AMGA services, available in the gLite, and we were able to include the estimation of several new physical quantities, which increased the total amount of data to be generated, stored, processed and visualized.

On the User Interface (UI) computer the scientist launches the Graphical User Interface (GUI) specially designed for this application. The job submission, monitoring and analysis of the results is controlled from there. The jobs are monitored from a monitoring thread, started from the GUI, and information about their progress is displayed to the user. Another thread run from the GUI is responsible for collecting the output results from the various Storage Elements

to the local one. For each output file a request for transfer is sent to the File Transfer Service (FTS) computer.

The *Web service* computer (WS) provides a grid-enabled secure gateway to the MySQL database. It accepts requests for new computations, distributes subtasks with the appropriate input parameters by requests from Worker Nodes and registeres successful computations and file transfers. The *AMGA* (ARDA Metadata Catalog) is used to hold information about the results obtained so far by the user - for example input parameters, number of jobs executed, execution date etc.

The WMS sends the job to the Grid sites. When the job starts on the WN (Worker Node), it downloads the executable from the Storage element. The executable obtains the input parameters from the WS, performs the computations and stores the results in the local Storage Element. It registers the output. One of the Worker Nodes is responsible for gradual accumulation of the output of the jobs. At regular intervals the accumulated results are registered and made available to the user.

The FTS is used in order to limit the number of files that are transferred simultaneously, because of the limited bandwidth available. In this way we also avoid some scalability limitations of the middleware and we try not to overload the Storage Elements. This approach is efficient, because in most cases it will not lead to increase of the total time necessary for completing all transfers, since they compete for the same network resource. Additional benefit of the FTS is that it provides reliable transfer of the files, by retrying the transfers if necessary.

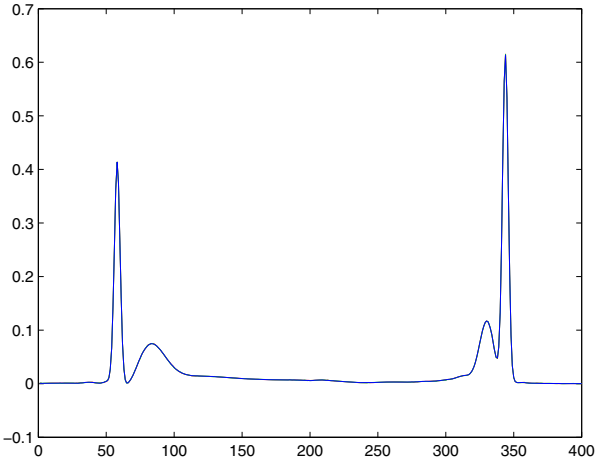


Fig. 1. Wave vector obtained with MCM and Hybrid1 (with Niedereiter) and Hybrid2 (with Halton) algorithm. The electric field is 15[kW/cm] along to the nanowire.

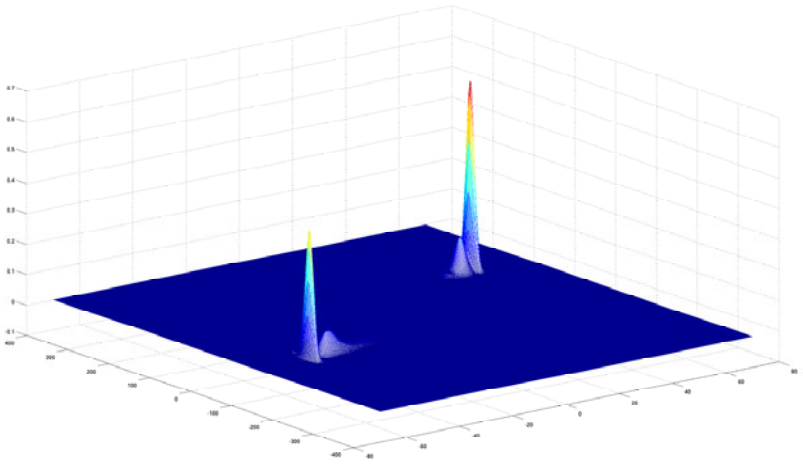


Fig. 2. The Wigner function at $140fs$ presented in the plane $z \times k_z$. The electric field is 15[kV/cm] along to the nanowire.

5 Numerical Tests and Grid Performance Analysis

The problems arising when we solve the Wigner equation using Monte Carlo approach are due to the large statistical error. This error is a product of two factors: standard deviation and sample size on the power one half. The standard deviation increases exponentially with time, so in order to achieve reasonable accuracy, we must increase considerably the sample size. This implies the need of computational resources.

Using the grid and above described grid implementation scheme, we were able to obtain new results about important physical quantities: Wigner function, wave vector, electron density and energy density. The results presented here are for inhomogeneous case with applied electric field for 140 femtoseconds evolution time. Normally, the execution times of the jobs at the different sites are similar, and the delay in starting is caused by lack of free Worker Nodes. Thus our new scheme allows the user to achieve the maximum possible throughput.

We have performed experiments with pseudorandom, and scrambled Niederreiter, Sobol and Halton sequences. In order to achieve the sufficient accuracy we have implemented 400 jobs (each with 4000000 trajectories) with our Monte Carlo algorithm, and 64 jobs (each with 2^{22} trajectories) with the hybrid algorithm (correspondingly, with scrambled Halton, Niederreiter and Sobol sequences). The mean square errors of $rez^{(i)}_{MCM} - rez^{(i)}_{Hybrid}$, where rez^i means wave vector, electron density energy density and Wigner function has order ranging from $O(10^{-4})$ to $O(10^{-5})$. But to achieve the *same results with the hybrid method we performed 6 times less trajectories.*

On the Figures 1 and 2 one can see the quantum effects - there is no symmetry when electric field is applied. The results obtained with MCM and the three

variants of the hybrid algorithm are plotted on the same picture for each of the estimated quantities. They are not visible on Fig. 2 because the error is very small. The best results are obtained using the Niederreiter sequence (in our version with the described scrambling algorithm). But the most important fact is that we achieved the same estimations with the hybrid method using 6 times less trajectories, i.e., six times less CPU time comparing to Monte Carlo.

References

1. Atanassov, E.: A New Efficient Algorithm for Generating the Scrambled Sobol' Sequence. In: Dimov, I.T., Lirkov, I., Margenov, S., Zlatev, Z. (eds.) NMA 2002. LNCS, vol. 2542, pp. 83–90. Springer, Heidelberg (2003)
2. Atanassov, E.I., Durchova, M.K.: Generating and Testing the Modified Halton Sequences. In: Dimov, I.T., Lirkov, I., Margenov, S., Zlatev, Z. (eds.) NMA 2002. LNCS, vol. 2542, pp. 91–98. Springer, Heidelberg (2003)
3. Atanassov, E., Gurov, T., Karaivanova, A., Nedjalkov, M.: Monte Carlo Grid Application for Electron Transport. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3993, pp. 616–623. Springer, Heidelberg (2006)
4. Atanassov, E., Gurov, T., Karaivanova, A.: SALUTE Application for Quantum Transport New Grid Implementation Scheme. In: Proceedings of Spanish e-Science Grid Conference, pp. 23–32, ISBN: 987-84-7834-544-1, NIPO: 654-07-015-9
5. Bromley, B.C.: Quasirandom Number Generation for Parallel Monte Carlo Algorithms. Journal of Parallel Distributed Computing 38(1), 101–104 (1996)
6. Caflisch, R.: Monte Carlo and quasi-Monte Carlo methods. Acta Numerica 7, 1–49 (1998)
7. Chi, H., Jones, E.: Generating Parallel Quasirandom Sequences by using Randomization. Journal of distributed and parallel computing 67(7), 876–881 (2007)
8. Chi, H., Mascagni, M.: Efficient Generation of Parallel Quasirandom Sequences via Scrambling. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4487, pp. 723–730. Springer, Heidelberg (2007)
9. Fischetti, M.V., Laux, S.E.: Monte Carlo Analysis of Electron Transport in Small Semiconductor Devices Including Band-Structure and Space-Charge Effects. Phys. Rev. B 38, 9721–9745 (1988)
10. Foster, J., Kesselmann, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Francisco (1998)
11. Gurov, T., et al.: Femtosecond Evolution of Spatially Inhomogeneous Carrier Excitations: Part II: Stochastic Approach and GRID Implementation. In: Lirkov, I., Margenov, S., Waśniewski, J. (eds.) LSSC 2005. LNCS, vol. 3743, pp. 157–163. Springer, Heidelberg (2006)
12. Kosina, H., Nedjalkov, M., Selberherr, S.: An event bias technique for Monte Carlo device simulation. Math. and Computers in Simulation 62, 367–375 (2003)
13. Nedjalkov, M., Kosina, H., Selberherr, S., Ringhofer, C., Ferry, D.K.: Unified particle approach to Wigner-Boltzmann transport in small semiconductor devices. Physical Review B 70, 115319–115335 (2004)

Mobile Agents for Management of Native Applications in GRID

Rocco Aversa, Beniamino Di Martino, Renato Donini, and Salvatore Venticinque

Second University of Naples, Aversa (CE), Italy
{rocco.aversa,salvatore.venticinque}@unina2.it,
renato.donini@gmail.com, beniamino.dimartino@unina.it

Abstract. Mobile Agents technology can be exploited to develop mobile Grid services. Here we present a Grid service for jobs management, implemented by Mobile Agents. Grid users can exploit the service by a WSRF interface being unaware about agents technology. A console has been designed to interface user applications with agents. Programmers are able to extend their applications with the ability to be check-pointed, suspended, resumed, cloned, dispatched. It can be done simply by adding some methods to their code, which specialize management on occurrence of particular events. We mean that applications do not need to be rewritten into different languages or adopting specific programming models. We realized a prototype implementation for management of native applications.

1 Introduction

We aim here at investigating how Mobile Agents technology can be used to develop advanced services for management of resources in distributed systems. Mobile Agents mechanisms such as autonomy, reactivity, clone-ability and mobility can be exploited for resource management and load balancing when system conditions change dynamically. Most of all mobile agents platforms are executed by Virtual Machines which make transparent the hardware/software architecture of the hosting node. It allows to distribute and execute mobile agents code on heterogeneous environments in a flexible way. On the other hand, most of legacy applications have been implemented by languages such as FORTRAN and C, and they are compiled for a target machine in order to optimize their performances. Here we present a mobile agent based service that allows for management of native applications on heterogeneous distributed systems. Agent technology has been used to provide management facility on any node where the execution of applications will be started. Programmers, in order to exploit the management service, can extend their application without modify the original code, but by overriding some methods which specialize the application life-cycle. We aim at supporting checkpoint, resume, migration and monitoring. Furthermore service is targeted to each Grid user, who is unaware about Agent technology and can exploit services facilities by a compliant WSRF interface. A console that allows

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 214–223, 2010.

the application control by an agent has been designed and implemented. An agent based service designed and implemented to automatically perform management strategies. In the second section related works on management services and load balancing mechanism are described. In section 3 we describe the facilities we have implemented in order to control the application life-cycle. In section 4 the software architecture of our service is presented. Section 5 provides an example of simple application that has been extended in order to exploit the service.

2 Related Work

Application management and migration are mechanisms developed in many environments for common purposes. There are many platforms which exploit migration to implement load balancing in distributed and parallel systems. When we deal with homogeneous clusters, process migration is supported to share resources dynamically and adaptively. MOSIX [1] provides a set of algorithms to react in real time to the changes of resources utilization in a cluster of workstations. Migration of processes is preemptive and transparent. The objective is to provide high performances to parallel and sequential applications. OpenMosix [2] is a decentralized version of MOSIX. Each node behaves as an autonomous system. Kerrighed [3] supports thread migration. When a node is less loaded a process is moved there from a more busy node. OpenSSI [3] does not need to save part of the process on the original hosting node. System calls are called locally. Not all the processes are candidates for migration.

On the other hand in heterogeneous systems process migration is not generally supported. Different environments are virtualized by a middleware that hides architectural details to the applications and supports portability. In this case is possible to migrate code and status of applications, but not to resume the process status. An hybrid approach that manage a Grid of homogeneous clusters is presented in [4] as an advance of research cited above. Some relevant contributions, which exploit Mobile Agents technology are [5,6]. We aim at exploiting flexibility of Mobile Agent programming to manage legacy applications without changing the original code and without rewriting the application into another language or adopting a new programming paradigm. Some approaches which should be compared with the one presented in this paper are cited below. [7] presents an approach to support mobility of applications in a GRID environment. A mobile agent is described by an UML-like language called blueprint. The blueprint description is transferred together its status. The receiving platform translates the agent blueprint description in a Java or a Python application that implement the original functionalities Blueprint is a language for a high level specification of agent functionalities and of its operational semantics. A specification describes the behavior of an agent in terms of its basic building blocks: components, control flow and data flow. Agent factories interpret the agent blueprint description and generate executable code composed of, for example, Java, Python, or C components. To support migration of common

user applications, [8] provides the possibility to insert same statements, such as go( ) or hop( ), between blocks of computation, such as for/while loops. A pre-compiler, such as ANTLR for C/C++ and JavaCC for Java source code, is used to substitute these statements with code that implements mobility. A user program may be entirely coded in C/C++ and executed in native mode. As a result Java agent starts the native code using a wrapper implemented by the JNI technology. In our approach programmers who want to support migration do not need to deal with any models, but they have just to handle such events which ask to save or resume the application status.

3 Management Facilities and Application Life-Cycle

We exploited Mobile Agents technology to allow services execution on heterogeneous platforms. We mean that we can execute monitoring and control facilities on heterogeneous nodes in a distributed environment. Furthermore we can move dynamically a service instance from a node to another resuming the execution at destination. Mechanisms of Mobile Agents technology have been adopted to design and implement advanced management facilities. Besides we provide the possibility to extend the same facilities to the managed application. Our model of application life-cycle is shown in Fig: 1. Regardless of the location, the process state could assume the following values: started, suspended, stopped and resumed. In the suspended mode, the application status has been saved in order to allow process restoring when a resume action will be invoked. Let us clarify that the application status is saved not the process one. That's because we aim at supporting mobility across heterogeneous architectures. Life-cycle can be monitored and controlled by a software console that generates and handles the following events:

1. *start*: starts the execution of a native application on a cluster node
2. *suspend*: suspends the native application saving its execution status.
3. *resume*: resumes the native application restoring the status saved on the last suspension.

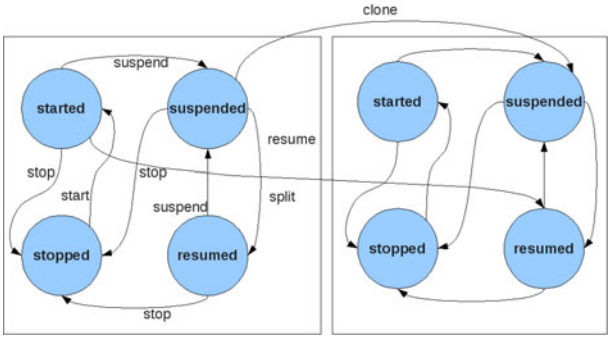


Fig. 1. Application life-cycle

4. *stop*: stops the native application.
5. *checkpoint*: saves the status of the native application without stopping its execution.
6. *migrate*: migrates the native application to a target node restoring its status at destination.
7. *split*: clones the native application and splits the job between the clones.

We have developed both an interactive GUI and a batch interpreter for submitting commands to the application console.

4 Service Architecture

As shown in Fig.2 software architecture of the management service is composed of different elements at different levels. A first set of components implements a portable service executed by Mobile Agents. Distributed Agents perform different roles. They implement user interface, application monitoring and control, code management. A monitor detects relevant changes of system conditions and notifies these events to the Agent Scheduler. The Agent Scheduler reacts in order to avoid performance degradation. It communicate and coordinates other agents. The Proxy Agent is responsible of the application execution. It receives requests from other agents and manages the application by a Java console. An abstract console defines the interface between the proxy and the application. It is obviously independent by the kind of application that will be controlled. Native applications will be linked to the console by mean of a native implementation of abstract methods. In order to support the execution on heterogeneous architectures, the programmer has to make available a new shared library that overrides native methods and that has been compiled for a target architecture. Libraries will be stored on a remote repository and the proxy agent is able to automatically download them, when the execution is moved to a new heterogeneous node.

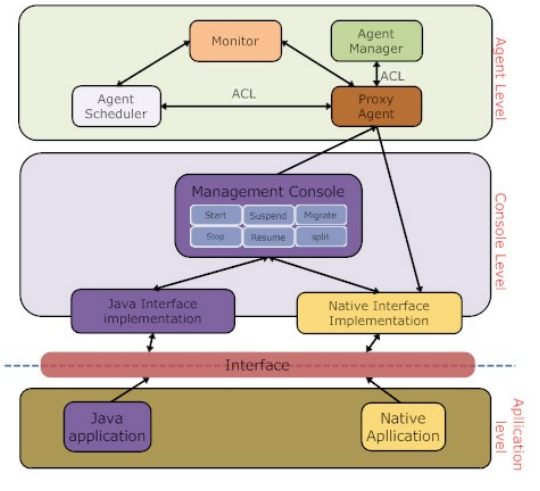


Fig. 2. Software Architecture

4.1 Agent Level

Agent Proxy. An Agent Proxy is delegated to manage an instance of users' application. It can:

- save and resume the status of the native application (save, resume);
- migrate, clone or split the application;
- handle special conditions such as termination.

For instance whenever the agent manager requires the migration of a native application to a selected node, the Agent Proxy has to transparently:

- suspend the native application as soon as possible;
- save the status of the native application;
- migrate to the target node;
- detect the target node hardware and software equipment;
- whenever it is necessary download and install the right library;
- restore the status of the native application;
- resume the native application.

Agent Manager. The Agent Manager provides a graphic interface for creating, migrating, cloning and destroying both Agent Proxies and the native applications. It sends standard ACL messages (ACl stands for Agent Communication Language) to ask for the actions should be taken by ProxyAgents. The Manager allows to load references to libraries which have been built and made available for different hardware and software architecture. It can handle independently multiple execution instances of the same application. A snapshot of the Management Console GUI is shown in Figure 3.

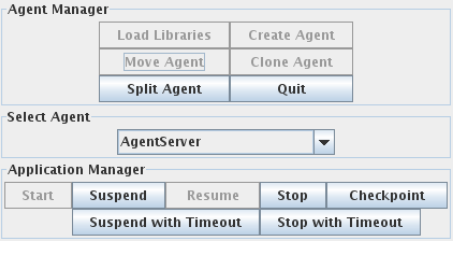


Fig. 3. Management Console Gui

Agent Monitor. Our architecture includes a monitor module to detect relevant changes of system conditions and notifies these events to the Autonomous Agent. As shown in the following the system is able to react to the events which affect the performance of the both service and application. The way to generate the events to send to the Batch Agent, could be less o more complex according to the particular requirements. Currently a simple configuration of management strategy based on threshold mechanisms is supported. The agent monitor checks the application's performance and compares it with a target input, such as the throughput of a web server; when a performance degrade was detected the right actions could be performed.

Agent Scheduler. The Agent Scheduler uses management facilities provided by Agent Proxies in order to carefully distribute the cluster workload or to optimize performance of applications. For instance, it can migrates native applications from a platform to another one that is less loaded. It can redistribute groups of applications in order to reduce network traffic by minimizing intercommunications, or reducing the overhead due to data transfer. Actually the user can configure the Scheduler behavior by associating a set of actions to a notification event. When a specific ACL messages has been received from the agent monitor the related batch file is interpreted and executed. A batch file can be written as described in Figure 4.2. In the example we show a sequence of commands which are executed on the occurrence of two events: *idle_node* and *busy_node*. Parameters of commands are extracted from content of related events notified by ACL messages. We are planning to adopt a more complete language such as the ones which support choreography or orchestration [9].

4.2 Console Level

In order to make transparent the management of different kinds of applications we defined the ApplicationManager abstract class. An implementation needs to support management of each kind of application. We provided a NativeManager class with java and native methods. Other classes implements special utilities. In particular:

- ApplicationManager: is an abstract class that represents an application manager.
- NativeManager: is a class that extends ApplicationManger It overrides abstract methods in order to interface with native applications.
- ApplicationThread: is a thread that starts native application and waits for events.
- Library: is a class that contains a set of parameters, which are used to identify and retrieve the compliant version of application library for the target machine architecture.
- ManagementException: is a class that implements an exception that could be thrown by the ApplicationManager.

JNI (Java Native Interface) is the technology that has been used to link the native implementation of the ApplicationManager to the Java implementation. JNI defines a standard naming and calling convention so the Java virtual machine can locate and invoke native methods. A native implementation of these methods have been developed in POSIX C and compiled for different architectures (AMD64, IA32, ...). As it is shown in Figure 4, in order to allow its application to be managed a programmer has to add those methods which override the ones defined in the native console. Linking the original application code with the new methods and the native console, a dynamic library for a target machine can be built. The native console is implemented by two POSIX processes: a father and its son. The son is spawn when the application starts. The father waits for requests from the Java part of the service, forwards them to the son that executes

```
<?xml version="1.0" encoding="UTF-8" ?>
<batch>
 <activation>
   <and>
     <event name="idle_node">
     <event name="busy_node">
   </and>
 </activation>
 <sequence>
  <operation>
   <command type="suspend" agent="$busy_node.agent_name[0]"/>
  </operation>
  <operation>
   <command type="move" agent="$busy_node.agent_name[0]">
    <parameters>
     <parameter>$idle_node.container[0] </parameter>
    </parameters>
   </command>
  </operation>
  <operation>
   <command type="resume" agent="$busy_node.agent_name[0]"/>
  </operation>
 <sequence>
</batch>
```

the application code by POSIX signals. The son before to start registers signal handlers and communicate by two POSIX pipes with the father. Pipes are used by the son to communicate to the father the file name where the application status has been stored and to communicate events such as a termination. Handlers can be specialized by the programmer by overriding the following methods (using C, FORTRAN, or any other native languages): *start_()*, *resume_(char*

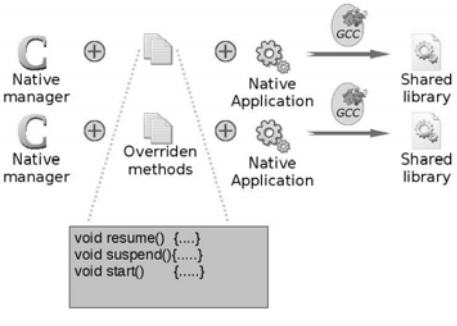
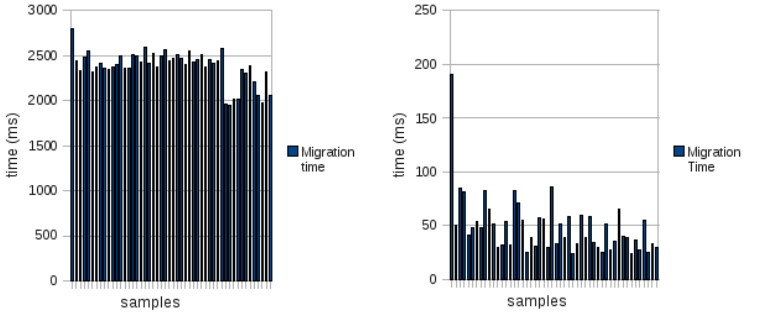


Fig. 4. Building a new library

status[]), *stop_(char status[])*, *suspend_(char status[])*, *checkpoint_(char status[])*, *split_(char status[])*, whose meanings has been described in the previous sections.

5 Case Study

We set up our Grid test-bed on an IBM blade cluster composed of 7 nodes, each of them with two Intel Xeon processors, 1 GB RAM, 72 GB hard disk, double (Giga) Ethernet network. The system is managed using a Rocks 5.1 Linux distribution. We chose, as case study, a POSIX C application to accomplish a brute force attack to drop an OTP like key. It is a computing intensive application with a regular behavior. This choice allows us to simplify our preliminary experiments. The application knows both plain text and cipher text. It ciphers plain text, and evaluates the result, using different set of keys till right one has been found. Cloning and migration can be exploited to distribute the workload on different nodes in order to improve the throughput of the application[10]. Succesfull experiments demonstrated that the prototype works properly, in fact the platform successfully supports checkpoint, resume, migration and monitoring of a native application. Results allowed us to evaluate the overhead due to cloning and migration when it is necessary to migrate the native code, which was previously compiled for the target machine.In this case migration overhead does not affect the application performance. However the migration time could be too high for certain kinds of applications. An analysis on the time required by each phase of the migration operation shows that the library downloading and installation is a rlelvant contribute to the relocation time as it can seen in Figure 5(a) while the size of the application status is not relevant in this case. Figure 5 shows the migration time when it is necessary to download the application library. In this case the agent suspends the application, migrates to a new node, downloads the library and the application status. In order to reduce the overhead we experimented a smart strategy for migration. A specialized agent



(a) Full migration time (b) Smart migration time

Fig. 5. (a) Full migration vs (b) smart migration

is sent to destination for downloading the library before to suspend the application. In this case the time elapsed between suspension and resume is reduced as it is can be seen in Figure 5 (b). Performance measures for a comparison of the two migration strategies are showed in the table.

	Full	Smart
min time	1942ms	24ms
max time	2792ms	189ms
average time	2376.89ms	49.2ms
accuracy	25.25ms	3,81ms

6 Conclusions

We presented a Mobile Agents based service for application management. Programmers can extend their native applications in order to support checkpoint, migration, suspension, resuming, etc. Service implementation is in Java. It is portable and mobile. Application portability on heterogeneous node is supported through the dynamic linking of native libraries compiled for the target machine. The application life cycle can be controlled interactively from a GUI or can be programmed by a scheduler that reacts to changes in the environment. An abstract console defines the methods which are used by agents in order to interface with the application. An implementation for POSIX application has been implemented and tested. Future work will deal with the design of automatic management strategies for distributing the workload in order to optimize system utilization or application performance.

References

1. Barak, A., La'adan, O.: The MOSIX Multicomputer Operating System for High Performance Cluster Computing. Journal of Future Generation Computer Systems 13(4-5), 361–372 (1998)
2. Bilbao, J., Garate, G., Olozaga, A., del Portillo, A.: Easy clustering with openMosix. World Scientific and Engineering Academy and Society (2005)
3. Lottiaux, R., Boissinot, B., Gallard, P., Valle, G., Morin, C.: openMosix, OpenSSI and Kerrighed: a comparative study, INRIA (2004)
4. Maoz, T., Barak, A., Amar, L.: Combining Virtual Machine Migration with Process Migration for HPC on Multi-Clusters and Grids. In: IEEE Cluster 2008, Tsukuba (September 2008)
5. Foster, I., Jennings, N.: Kesselman, Brain Meets Brawn: Why Grid and Agents Need Each Other. In: Proceeding of the 3rd Joint Conference Autonomous Agents and Multi-Agent Systems, pp. 8–15 (2004)
6. Di Martino, B., Rana, O.F.: Grid Performance and Resource Management using Mobile Agents. In: Getov, V., et al. (eds.) Performance Analysis and Grid Computing, pp. 251–264. Kluwer Academic Publishers, Dordrecht (2004)
7. Brazier, F.M.T., Overeinder, B.J., van Steen, M., Wijngaards, N.J.E.: Agent Factory: Generative Migration of Mobile Agents in Heterogeneous Environments. In: Proceedings of the 2002 ACM Symposium on Applied Computing, pp. 101–106 (2002), ISBN:1-58113-445-2

8. Fukuda, M., Tanaka, Y., Suzuki, N., Bic, L.F., Kobayashi, S.: A mobile-agent-based PC grid. In: Autonomic Computing Workshop, pp. 142–150 (June 2005), ISBN: 0-7695-1983-0
9. Peltz, C.: Web Services Orchestration and Choreography. IEEE Computer Magazines (2003)
10. Donini, R., Aversa, R., Di Martino, B., Venticinque, S.: Load balancing of mobile agents based applications in grid systems. In: Proceedings of 8 IEEE 17th Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Rome, June 23-25. IEEE, Los Alamitos (2008)

Leveraging Complex Event Processing for Grid Monitoring*

Bartosz Balis[1,2], Bartosz Kowalewski[2], and Marian Bubak[1,2]

[1] Institute of Computer Science, AGH, Poland
{balis,bubak}@agh.edu.pl, kowalewski.bartosz@gmail.com
[2] Academic Computer Centre – CYFRONET, Poland

Abstract. Currently existing monitoring services for Grid infrastructures typically collect information from local agents and store it as data sets in global repositories. However, for some scenarios querying real-time streams of monitoring information would be extremely useful. In this paper, we evaluate Complex Event Processing technologies applied to real-time Grid monitoring. We present a monitoring system which uses CEP technologies to expose monitoring information as queryable data streams. We study an example use case – monitoring for job rescheduling. We also employ CEP technologies for data reduction, measure the overhead of monitoring, and conclude that real-time Grid monitoring is possible without excessive intrusiveness for resources and network.

Keywords: Grid computing, real-time monitoring, event-driven architecture, complex event processing, event correlation.

1 Introduction and Motivation

Monitoring services are integral part of large scale Grid infrastructures. Typically, monitoring activities focus on reporting the current status and utilization of resources, and gathering historical data in order to enable retrospective analysis. Monitoring information is usually collected locally and disseminated to a site-level or central server where it is stored, refreshed periodically and exposed for querying by consumers. However, in certain cases a more real-time access to *monitoring information streams* would be desired. Examples of these include SLA contract monitoring, real-time system misuse detection, failure detection, or real-time monitoring of resource utilization for the purpose of steering and adaptive algorithms, such as job rescheduling [4]. Given the dynamic nature of the Grid which is characterized by variable resource demands and dynamic application behavior, this type of monitoring is particularly important.

Complex Event Processing (CEP) [10] is a general term that describes all approaches that take streams of atomic events, enable querying over those streams, and produce derived complex events. Nowadays advanced event processing mechanisms [6] are being introduced and CEP engines are capable of discovering

* This work is supported through Polish PL-Grid Project, AGH grant 11.11.120.865, and ACC CYFRONET AGH grant 500-08.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 224–233, 2010.

sophisticated patterns in the event stream. Surprisingly, though monitoring information can be viewed as streams of data reflecting current status and happenings within the Grid infrastructure, CEP technologies have not been employed to build Grid monitoring services.

The goal of this paper is to evaluate the Complex Event Processing technologies as a basis of Grid monitoring services. We have built a Grid monitoring infrastructure – GEMINI2 – which uses a CEP engine Esper to provide monitoring information as real-time streams [12]. We show that CEP technologies benefit Grid monitoring services as in several ways: (1) Enable to expose monitoring information as queryable data streams accessible in real-time; (2) Provide highly expressive querying constructs and high-performance engines that enable such capabilities as filtering, aggregation, sliding window calculations, and event correlation; (3) Enable data reduction based on buffering, filtering and aggregation. We study a use case – monitoring used for job rescheduling in the Grid, and evaluate the overhead of monitoring services on the Grid resources.

This paper is organized as follows. Section 2 presents related work. The GEMINI2 infrastructure is described in section 3. Section 4 overviews the current implementation of GEMINI2. In section 5, GEMINI2 is evaluated, including the case study scenario, and monitoring overhead evaluation. Section 6 concludes the paper.

2 Related Work

In existing Grid monitoring and information systems, monitoring information is typically collected by local sensors and disseminated to a site-level or global repository, subsequently queried by consumers. A representative Grid monitoring service which adopts those design principles is GridICE [1], a system used within the EGEE project. In GridICE, sensors collect monitoring information about local resources and disseminate it to a site collector where it is converted and stored according to a data model – the Glue schema [2]. The monitoring information can be collected from individual sites through the Grid information system interface, and aggregated at the global level by a GridICE server.

A similar architecture is featured by Inca 2 [11], a monitoring tool used in the TeraGrid project. Inca 2 focuses on detection of problems in the Grid infrastructure. To this end, testing of Grid software and services is periodically performed by local agents, called *reporters*, which are managed by *reporter managers*. The results of the tests are stored in a *depot*, which can be queried by consumers. Several other existing Grid monitoring systems, which cannot be described here because of space limitations, adopt similar design principles.

R-GMA [5] is to some extent similar to our approach in that it views monitoring information as streams published by producers and requested by consumers via distributed queries. As R-GMA adopts a relational model for monitoring information, the streams are tuples (table rows), requested by consumers via SQL queries. However, SQL and the relational model impose several restrictions onto processing and querying over data streams. On the other hand, CEP technologies have the advantage of being specifically designed for this purpose. In CEP,

unlike in SQL / relational model, features such as aggregation, sliding window calculations or correlation are naturally available.

In summary, existing Grid monitoring services are not oriented towards enabling real-time subscriptions for monitoring information. Most solutions do not expose monitoring information as queryable data streams, but convert it to data sets stored in a permanent repository. While this is useful for certain scenarios (e.g., those where historical data is needed), it is not well-suited for real-time querying. For example, temporal aspects of data streams are lost during conversion, along with querying capabilities that rely on it (such as sliding window calculations or correlations).

The work described here is a result of our previous experience in event-driven systems [8] and monitoring in the Grid based on an earlier non-CEP based monitoring system GEMINI [3].

3 Concept of GEMINI2

Generic Monitoring Infrastructure (GEMINI2) is a lightweight framework designed to provide event-based mechanisms for distributed environments. Though we demonstrate its use for monitoring, it can provide event-based mechanisms for any distributed system.

3.1 Requirements

A few important assumptions were made in the initial stages of the project that had a significant impact on the final design of the framework. These prerequisites were to clearly draw a distinction between the currently available solutions and set of features expected to be provided by GEMINI2. Amongst the main requirements that were identified at that stage are usability, configurability, performance, scalability, and standards-based approach.

GEMINI2 as a generic monitoring infrastructure also needs to define a taxonomy for events passed through the system and provide a standard way of representing these events as event objects. The monitoring events' hierarchy will be based on already available documentation that attempted to summarize the set of monitoring events currently used in Grid environments. There are several papers and memos that cover this subject. Nevertheless, event types used in monitoring measurements haven't been standardized yet.

3.2 Leveraging Complex Event Processing

Applied to monitoring, CEP enables real-time processing of monitoring information streams, including among others: (1) aggregation of smaller events in order to provide a high-level view of a process – statistics, summaries, etc.; (2) correlation of events generated by different event sources; (3) long-term metrics/measurements.

A CEP engine incorporated into a monitoring infrastructure not only provides powerful stream querying capabilities and discoverability, but also enables data

reduction, essential to avoid network flooding. Section 3.4 describes this aspect in more detail.

GEMINI2 aims at providing interchangeable building blocks that will significantly simplify deploying a monitoring (or, more generally, eventing) infrastructure based on CEP, in a distributed environment. GEMINI2 exposes event dissemination and management interfaces using standard communication technologies while delegating the responsibility of identifying complex situations to an exchangeable CEP engine. The initial versions of the infrastructure are built upon Codehaus Esper, a popular and powerful open-source CEP solution.

Consequently, the interface to request monitoring information is based on Event Processing Language (EPL), used by Esper. EPL is a declarative language for expressing queries over event streams, using an intuitive SQL-like syntax. The drawback of EPL is that it is not an industry standard. Supporting the process of CEP standardization is one of the future goals of GEMINI2.

3.3 Design

In order to support configurability and usability, GEMINI2 infrastructure was divided into a group of interchangeable components that could be used to easily assemble a solution suitable for any particular distributed environment.

Fig. 1 presents a simplified view over the architecture of GEMINI2. At the same time this diagram depicts a standard deployment configuration to be used in a distributed environment instrumented using GEMINI2.

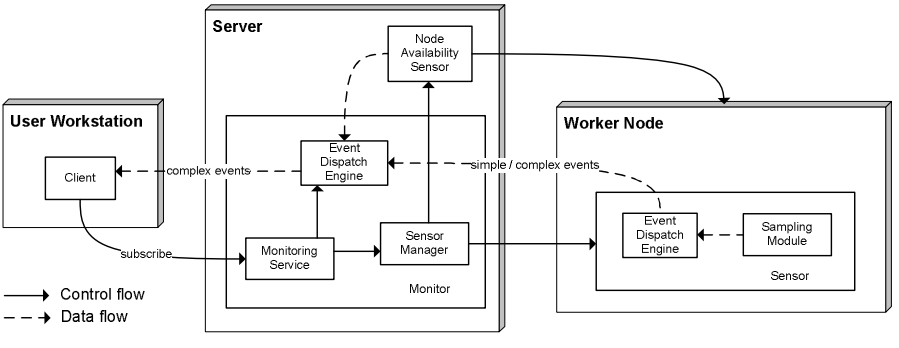


Fig. 1. High-level view over GEMINI2 architecture and a sample deployment of its components

There are three main logical entities that build every distributed monitoring environment: Sensors, Monitors and Clients. Sensors are responsible for sampling the environment (nodes, network, etc.) and generating simple events. They can also incorporate an Event Dispatch (CEP) engine in order to apply Complex Event Processing mechanisms directly to stream of events generated during the sampling process. This way event objects disseminated by the sensor are already preprocessed in order to decrease data volume.

Monitors are responsible for handling subscription-related control messages coming from Clients and processing high volumes of event objects received from Sensors. The incoming events are passed to an Event Dispatch (CEP) engine which contains definitions of complex queries associated with particular client subscriptions. New event streams produced by the Dispatch Engine are then disseminated to proper Clients.

Monitors can be used to build complex topologies of cooperating server nodes. This makes the deployment architecture even more flexible and scalable.

Clients subscribe to monitors for particular complex events. Such control messages as subscribe, renew subscription and unsubscribe are passed to the Monitoring Service running inside a Monitor. Clients then accept event objects disseminated by the Monitor through a separate communication channel.

Each of the three constituent parts of the infrastructure is assembled using exactly the same set of reusable components. GEMINI2 employs an Inversion of Control (IoC) container in order to enable one to easily create their own infrastructure setup. The common set of components includes event dissemination engines, monitoring service subparts and stubs, Event Dispatch Engine, Web Services transport endpoints for control messages, JMS transport endpoints for event channels, and many more.

3.4 Data Reduction

Data reduction is important in the Grid in order to avoid excessive network overhead due to monitoring. Complex Event Processing constructs naturally enable one to achieve data reduction by buffering, filtering or aggregation, for example:

- `select * from CpuInfoMsg(idletime<0.5)`
  Send $CpuInfoMsg$ events, but only those where CPU idle time drops below 0.5.
- `select * from CpuInfoMsg.win:time_batch(5 sec)`
  Notify collected events in one batch every 5 seconds.
- `select avg(cpu.idletime) from CpuInfoMsg.win:time(3 min) as cpu having avg(cpu.idletime)<0.5 output last every 30 seconds`
  Compute average of a CPU idle time over a 3 minutes time window, and report it once every 30 seconds, but only when it drops below 0.5.

The most important principle for data reduction is to process events as close to their source as possible. Therefore, we employ the CEP engine also at the sensors' side.

Sensors which measure a certain quantity by sampling are characterized by a certain sampling rate, for example CPU idle time may be measured once per 30 seconds. The sampled measurements are reported as event stream to the local sensor's CEP engine. A Monitor may subscribe to this stream for the most accurate and fine-grained measurements. However, for any measurement, additional streams may be defined which expose the measurement in a more aggregated form.

Let us consider an example of $CpuInfoEventMessage$ measurement which, among others, delivers the current idle time of a CPU. In order to subscribe for the event stream representing this measurement, a Monitor may simply send the following request to a sensor: `select * from CpuInfoEventMessage`. This will result in sending events at the sensor's sampling rate. However, a sensor may be configured to also internally activate the following subscription:

```
insert into CpuInfoAvg select avg(cpu.idletime) as idletime
from CpuInfoEventMessage.win:time(5 min) as cpu
output last every 5 minutes
```

This request simply consumes the $CpuInfoEventMessage$ stream, computes an average of the *cpu.idletime* attribute over a 5 minute window and reports this average once per every 5 minutes. The EPL `insert into` construct inserts the resulting event stream back to the engine, so that new $CpuInfoAvg$ stream is available for subscription by consumers. The expression `as idletime` simply ensures that the attribute names in the output $CpuInfoAvg$ stream are the same as in the original one. As a result, the two streams can be used in the same way. The Monitor, by subscribing to the $CpuInfoAvg$ stream receives one aggregated event per 5 minutes, instead of one per every 30 seconds.

4 Implementation

The infrastructure is implemented in Java, making it naturally suitable for Java-based applications. Nevertheless, GEMINI2 is based on widely accepted standards, which make the infrastructure interoperable. Main control interfaces defined by Monitoring Service are exposed using Web Services (SOAP over HTTP). Event dissemination is based on JMS (ActiveMQ). Support other messaging technologies is also planned. The complexity of used technologies is hidden to end users behind standard building blocks and intuitive interfaces.

Standards upon which GEMINI2 is built include Web Services (SOAP over HTTP), WS-* (WS-Security) being planned to be supported in future versions of the system; XML, JAXB; JMS; Google Protocol Buffers; REST (planned to be supported in future versions of the system).

Technologies employed in the infrastructure include Apache CXF; Spring IoC container; Apache ActiveMQ; Codehaus Esper CEP engine (foundation for the first version of the Event Dispatcher engine); SIGAR (Hyperic's System Information Gatherer) for collecting data from the monitored nodes.

5 Evaluation

5.1 Case Study Scenario

Let us consider a representative scenario in which the benefits of using CEP for monitoring can be observed – job rescheduling. It has been pointed out that for complex Grid applications, such as workflows [7] which can be described as

DAGs of tasks, static scheduling strategies are not optimal. Therefore, dynamic scheduling is employed in which dynamic information about resource usage and application progress is used in order to dynamically change the mapping of tasks to resources. In a typical scenario, when a node currently listed in a schedule displays a prolonged performance degradation or a failure, a rescheduling algorithm is triggered which attempts to find a better candidate. A solution for monitoring for this scenario involves real-time monitoring of event streams, filtering, aggregation, and event correlation – all provided by CEP technologies.

Fig. 1 presents a sample deployment of monitoring components for a single node monitored by a scheduler. A Worker Node (WN) is assigned in the current schedule for a task of a workflow job to be executed. A sensor deployed on the WN monitors the current capacity of the CPU (measured as its idle time). At the same time, a Node Availability Sensor deployed on a remote server periodically checks whether the node is available. Let us assume that the scheduler requires a notification if the average CPU availability measured over last 3 minutes drops below 70%, or the node fails (becomes unavailable). When either happens, the scheduler will attempt to find another candidate node for the task in question.

The following two EPL statements are sufficient to create a monitoring information stream required by the scheduler (the statements are simplified in that specific resource identifiers are omitted):

```
insert into AvgCpu3 select avg(idletime) as avgCpuIdle from
CpuInfoEventMessage.win:time(3 min)
having avg(idletime)<0.7 output last every 30 seconds

select cpu.idletime as cpuIdle, node.avail as nodeAvail from
pattern [every (cpu=AvgCpu3 or node=NodeInfoMsg(avail=0))
```

The first statement consumes the $CpuInfoEventMessage$ stream and creates a new event stream $AvgCpu3$ which computes average CPU idle time over a 3 minute time window and reports the measurement every 1 minute, but only if the computed average drops below 0.7. In reality, this measurement would be more complex given that currently nodes feature many CPUs and cores. However, it would not be difficult to generalize this request for multiple cores.

The second statement selects properties from two event streams – $AvgCpu3$ and $NodeInfo$, and returns events that contain information about CPU capacity or node availability, but only if the conditions required by the scheduler are met.

5.2 Monitoring Overhead

The approach described in this paper attempts to prove that network and CPU load caused by a monitoring infrastructure does not necessarily have to be huge. We conducted several experiments that were to clearly show that if configured properly, GEMINI2 only adds little overhead. Monitors and sensors do not influence nodes' operation and do not introduce the risk of performance degradation.

For tests, sensors were deployed on 5 nodes (40 cores) of the CYFRONET's *Zeus* cluster[1], while a monitor was deployed on `ui.cyf-kr.edu.pl`. The tests only covered channels used to disseminate events, excluding Web Services-based control channels where traffic is much lower.

Three scenarios were evaluated: *Raw*, *Aggregated* and *Buffered*, described in Table 1. A client deployed on `ui.cyf-kr.edu.pl` requested a simple EPL query:

```
select avg(cpu.idletime) from
       CpuInfoEventMessage.win:time(3 min) as cpu
       output last every 3 minutes
```

During the tests we measured CPU utilization and memory consumption for the monitor process and for one of the sensors. We also analyzed network traffic between the monitor and sensors. CPU utilization levels were similar in all scenarios. With only 5 sensors deployed the Monitor process did not consume a lot of CPU time. In all our experiments CPU usage was rounded to 0.0%. Sizes of the processes were also the same in all of the configurations. Table 1 presents network traffic measurements. Each of the experiments lasted around a day.

Table 1. Overhead test scenarios and network overhead measurements

Scenario No.	Description	Avg traffic (KB/min)
1: Raw	Sensor samples its CPUs every 2 sec and disseminates 8 events (one per core) immediately	161.5
2: Aggregated	Sensor samples CPUs every 2 sec, computes average over 30 measurements and disseminates 8 events every minute	6.3
3: Buffered	Sensor samples CPUs every 2 sec and disseminates $30 * 8$ events in a batch every minute	153.0

We can clearly see that the network overhead varies depending on the configuration being used. If CEP-based mechanisms are introduced close to the sensor, the traffic is significantly reduced.

GEMINI2 uses a high compression rate for the events being disseminated between Sensors and a Monitor, and between a Monitor and clients. Google Protobuf which is used to marshal objects ensures that data is packaged optimally. On the other hand, wire format employed by Message Oriented Middleware (MOM) implementation used as a transport for event objects (Apache ActiveMQ) is far from being optimal. It adds substantial overhead, both when event object is disseminated and the message is acknowledged, and also when executing standard messaging-related tasks, i.e. establishing JMS connections, JMS sessions, JMS consumers, sending keep-alive packets, etc.

Simple tests were conducted in order to measure overhead added by ActiveMQ. Wireshark was used to capture TCP traffic and provide overall size

[1] HP Cluster Platform 3000 BL 2x220 – 256 2-CPU, 4-core 'blade' nodes, Intel Xeon Quadcore 2.5 GHz, 2GB RAM per core.

of data sent over the wire. The experiment showed that sending two subsequent events with CPU usage information caused between 20 and 30 packets to be sent. The overall size of information sent reached 1,5 KB, while packets used to carry event objects and acknowledge them had overall size of 0,4 KB. What is more, the average size of single Protobuf-encoded CPU usage event is around 0,1 KB, so only 0,2 KB out of 1,5 KB of data was meaningful from the point of view monitoring.

The experimental data clearly shows that the overhead added by GEMINI2 is small. If properly configured, GEMINI2 components minimize data exchange via a significant data reduction. Moreover, we showed that the main source of the network overhead are the technologies used to transmit event objects. In future we may consider optimizing wire format used by ActiveMQ or evaluating other MOM solutions. Our experiments also showed one potential risk of using GEMINI2. The infrastructure provides one with extremely powerful mechanisms and at the same time it is highly configurable. Low overhead observed in our environments was a consequence of proper configuration of GEMINI2 components. Incorrectly configured GEMINI2 can easily lead to an excessive network utilization.

6 Conclusion

We have evaluated Complex Event Processing technologies for real-time monitoring of the Grid infrastructure. We have presented GEMINI2 – a Grid monitoring system which leverages Complex Event Processing technologies in order to expose monitoring information as queryable event streams. It was shown that CEP technologies enable high-performance, real-time access to monitoring information, at the same time providing means to data reduction so that excessive intrusiveness due to monitoring can be substantially reduced.

We have found that the EPL stream query language provides expressiveness sufficient to support complex scenarios, not achievable through traditional database query languages. The Esper engine, on the other hand, enables high performance event processing.

Future plans for the GEMINI2 project include, amongst all the other goals, experimenting with various CEP-related technologies and approaches. All the possible ways of creating CEP rules need to be evaluated, including, but not limited to, XML-based, SQL-based and SQL-like queries. A possibility of creating a new custom CEP engine dedicated for the monitoring use cases is also taken into account. One of the goals of GEMINI2 is also to support the process of standardization of Complex Event Processing mechanisms and languages. An attempt to improve event processing formalisms will be made as a part of the mentioned standardization process. We also plan to expand our tests in a real Grid environment to a much larger number of nodes, extend measurements to new objects, including Grid jobs, and conduct experiments with highly complex queries.

Acknowledgements. We would like to thank Patryk Lason for his help when configuring Grid environment and conducting experiments.

References

1. Andreozzi, S., Bortoli, N.D., Fantinel, S., Ghiselli, A., Rubini, G.L., Tortone, G., Vistoli, M.C.: GridICE: a monitoring service for Grid systems. Future Generation Computer Systems 21, 559–571 (2005)
2. Andreozzi, S., Sgaravatto, M., Vistoli, M.C.: Sharing a conceptual model of grid resources and services (2003)
3. Balis, B., Bubak, M., Pelczar, M.: From Monitoring Data to Experiment Information – Monitoring of Grid Scientific Workflows. In: Third IEEE Int. Conference on e-Science and Grid Computing, e-Science 2007, pp. 187–194. IEEE Computer Society, Los Alamitos (2007)
4. Berman, F., et al.: New grid scheduling and rescheduling methods in the GrADS project. Int. J. Parallel Program. 33, 209–229 (2005)
5. Cooke, A., Gray, A., et al.: R-GMA: An Information Integration System for Grid Monitoring. In: Meersman, R., Tari, Z., Schmidt, D.C. (eds.) CoopIS 2003, DOA 2003, and ODBASE 2003. LNCS, vol. 2888, pp. 462–481. Springer, Heidelberg (2003)
6. Faison, T.: Event-Based Programming: Taking Events to the Limit, ch. 1. Apress (2006)
7. Gunter, D.K., Jackson, K.R., Konerding, D.E., Lee, J.R., Tierney, B.L.: Essential Grid Workflow Monitoring Elements. In: International Conference on Grid Computing and Applications. CSREA Press (2005)
8. Kowalewski, B., Bubak, M., Balis, B.: An Event-Based Approach to Reducing Coupling in Large-Scale Applications. In: Proc. Computational Science - ICCS 2008, 8th International Conference, Part III, Kraków, Poland, June 23-25 (2008)
9. Legrand, I.C., Newman, H.B.: MonALISA: An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications. In: CHEP 2004 (2004)
10. Luckham, D.C., Frasca, B.: Complex Event Processing in Distributed Systems. In: Program Analysis and Verification Group, Computer Systems Lab. Stanford University (1998)
11. Smallen, S., Ericson, K., Hayes, J., Olschanowsky, C.: User-level grid monitoring with Inca 2. In: GMW 2007: Proceedings of the 2007 Workshop on Grid Monitoring, pp. 29–38. ACM, New York (2007)
12. Wu, E., Diao, Y., Rizvi, S.: High-Performance Complex Event Processing over Streams. In: 2006 ACM SIGMOD Int. Conference on Management of Data (2006)

Designing Execution Control in Programs with Global Application States Monitoring

Janusz Borkowski[1] and Marek Tudruj[1,2]

[1] Polish-Japanese Institute of Information Technology,
86 Koszykowa Str., Warsaw, Poland
[2] Institute of Computer Science, Polish Academy of Sciences,
21 Ordona Str., Warsaw, Poland
{janb,tudruj}@pjwstk.edu.pl

Abstract. The paper is concerned with methodology for designing ececution control based on application global states monitoring in parallel and distributed programs. The principles of global state monitoring i parallel programs are recalled. Then new control statement semantics related with the global states monitoring are proposed, including synchronous and asynchronous approach to the maintenance of information used for the global states-control in programs. Proposals of syntactical solutions for designing a control language which accounts for program global states in execution control in parallel/distributed programs are presented.

1 Introduction

It has been commonly noticed that execution control in parallel and distributed programs is inherently composed of two constituent components: local control embedded inside host processors at the level of processes or threads and global control that is defined based on information on global parallel states in many constituent processors. Currently existing parallel programming message passing environments provide a programmer only with very basic primitives to express program execution control in parallel programs. A programmer has to design by himself the necessary inter-process communication and to include into parallel processes, so that the knowledge of the computation parallel state by constituent processes could enable to implement the desired control in the programs at the global application level. Designing such control is complicated and error-prone if the global control conditions should account for states of many distributed processes. Additionally existing parallel programming methods usually mix the main computation code with control statements. With such a practice, the logic, which supervises execution of a parallel application, is difficult to maintain.

The approach discussed in this paper applies the notion of the global states and predicates computed on global states in the distributed systems for the on-line control of programs execution. The control flow conditions in programs are decoupled from the program code as predicates embedded in well defined syntactic program elements, which are easily identified, understood, verified and

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 234–243, 2010.

modified if necessary. The global states related control primitives can be directly applied for program correctness verification in an analogous way to distributed debuggers. The control directly based on global control predicates can also make the programs correct by construction. The notion of global application states has been known for a long time and used in practice concerned with parallel and distributed program debugging. Earlier works e.g. [1] provided methods for capturing a global state which is consistent. Later the research was more focused on obtaining a whole consistent global state space and checking its properties [2,3]. The checking was aimed at detecting unwanted system behavior (monitoring for faults) and ensuring proper system operation.

In [4] a system has been proposed, in which control statements in process code could take into account states of other processes. The states were expressed as values of special variables replicated among processes in defined intervals. Our work can be seen as enhancing this work by using global consistent states theory to provide a sound framework for global application control.

Our approach was to develop a high-level synchronization method, allowing for separation of the control logic from computational code, and providing the programmer with a global view of the state of a parallel application, even in distributed environments. We proposed a novel framework for control of parallel/distributed applications, which clearly separates the global control subsystem responsible for application control from a local process code, and which detaches control network traffic from bulk data exchange [5]. Processes report their local states to a monitor. A monitor hierarchy can be used if a single monitor does not perform adequately, so even large systems can be controlled with the proposed method [6]. A monitor observes application global states constructed from received local states and evaluates global predicates. The results of the evaluation influence the control flow of application processes.

A similar approach has been implemented in the PS-Grade system, based on graphical design of parallel programs. PS-Grade has been developed as an enhancement to an existing graphical parallel programming system (P-GRADE). It supported interesting but limited functionality in the area of application global control – application state on-line monitoring combined with asynchronous reactions using the theoretical approach described in point 3.2. The PS-Grade project will be continued by designing a new control framework with more general approach to internal control design based on global states monitoring. Influencing program execution control by global states monitoring can be implemented using a number of ways, including the different methods of global states observation, extending their morfology, defining global states representation and respective predicates functionality, the structure of the control traffic embedding and supervising and finally the different types of direct affecting the behaviour of application program components. This work is an introductory work concerned with designing such more generalized control approach.

The paper is composed of 3 main parts. In the first part the principles of global application state monitoring are recalled. In the second part the control statement semantics related with the global states monitoring are discussed, including

synchronous and asynchronous reactions to global states control predicates in programs. The third part presents some syntactical solutions for designing a control language which accounts for global states in execution control in parallel/distributed programs.

2 Global Application State Monitoring

A monitor observing a running application gets local state reports from application processes. The reports are transferred by means of message passing to be easily implementable in distributed environments. The sequence of reports as seen by a monitor may not be the same as a sequence of events in reality, because delays in message construction, transfer and processing can occur. Therefore, the received messages can be used only as input data for consistent state construction algorithms [7]. A consistent global state is a global state consisting of process local states which are pair wise concurrent.

Usually logical vector clocks are used as event timestamps allowing for concurrency verification. Global state algorithms working with logical time are very costly or they can deal with a restricted class of global predicates only. Alternatively, real-time timestamps obtained with the use of partially synchronized local process clocks can be employed to construct Strongly Consistent Global States (SCGS) [8]. A SCGS is a state, which has really occurred (as opposed to potential global states obtained with the use of logical clocks) and could be discovered by a monitor by analyzing received reports about process events annotated with timestamps of assumed uncertainty. Event report processing and consistent states construction delays control decisions. Therefore, we use consistent states only when they are necessary.

3 Control Statement Semantics for Global States Monitoring

The presented synchronization method based on global predicate evaluation assumes, that predicate fulfillment is communicated to application process. The processes should react accordingly. Further control implementation issues such as the way in which processes receive information about predicate fulfillment and the method for programming the reaction on predicate fulfillment will be discussed in next sections.

3.1 Synchronous Global Control Statements

Synchronous global control statements are a part of the normal process sequential code. They are executed when the execution point reaches them. We propose using global control statements $g(\varphi)$, e.g. $if(\varphi)$ or $while(\varphi)$, where φ is a global predicate defined at a monitor. A similar, but simpler idea was proposed in [4], without a notion of global states, proposing shared replicated variables and phased program execution with replication taking place between phases.

In the discussed approach, the parallel application control is composed of the following components:

1. a way of expressing process states e.g. values of chosen variables,
2. global predicates definitions, created at a monitor to reflect the desired synchronization pattern,
3. modality of global predicates, defining on what global states the predicates are evaluated, e.g. SCGS as described in the previous section,
4. a way of choosing a particular global state, on which predicate φ is evaluated in a global control statement $g(\varphi)$,
5. code of processes using synchronous global control statements $g(\varphi)$.

Point 4 requires more in-depth considerations. We propose the following alternative methods to determine the concrete global state used for global predicate evaluation:

fully synchronous – a synchronous global control statement is present in the code of all parallel processes composing an application. Upon reaching the global control statement in a process, the execution suspends until all the other processes reach the same control statement. The global state used for predicate evaluation is composed of local states existing at the moment of reaching the global control statement. This idea resembles predicate modality *currently* [3].

phased execution – the previous proposition requires the processes arriving early at a global control statement to wait until all other processes reach this statement. To get rid of waiting, the predicates may use some historical – already known – global state. The program execution can be split into phases. Then, to evaluate predicates in phase n, we should take into account the last state from phase *n-1*. One can notice here a resemblance to phased data transfers in BSP model [9].

snapshot – a snapshot represents a single global state of a parallel application obtained by running a special distributed snapshot algorithm. A process executing a global control statement can initiate a snapshot algorithm, e.g. [1], to obtain a global state on which a predicate can be evaluated. Process execution must be suspended until the snapshot is ready. In this case no monitor is necessary. However, the obtained snapshot is not deterministic (because the considered system is not synchronous and the clocks are not ideally synchronized), so this method may be used to test stable predicates only. A stable predicate has a property, that once it becomes fulfilled it stays so forever.

the current state – a process executing a global control statement evaluates the global predicate on a global state containing its current local state. Many such global states can exist. We distinguish three ways of choosing a single one:

- *earliest* – a global control statement $g(\varphi)$ in process P_i is evaluated on the earliest global state containing the current local state of P_i . Such a global state is uniquely defined if SCGS are used. In Fig. 1, the global predicate is evaluated on global state S_1 if *earliest* mode is applied.

- *recent* – a global control statement $g(\varphi)$ in process P_i is evaluated on a recently known (latest) global state, containing the current local state of P_i . When using the recent mode in Fig. 1, the global predicate is evaluated on global state S_1, S_2 or S_3, depending on which events from S_1 were known (reports about them have arrived) just when the evaluation is requested. This mode uses the latest available state information, but does not select global states deterministically.
- *moment* – a global control statement$g(\varphi)$ in process P_i is evaluated on a global state comprising the moment of reaching $g(\varphi)$ by P_i. To be able to pinpoint such a moment, real time timestamps and SCGS can be used. In Fig. 1, the global predicate in the moment mode is evaluated on the global state S_2 . However, there can exist no SCGS comprising the moment, when the process execution reaches $g(\varphi)$. Then, the first following global state should be used.

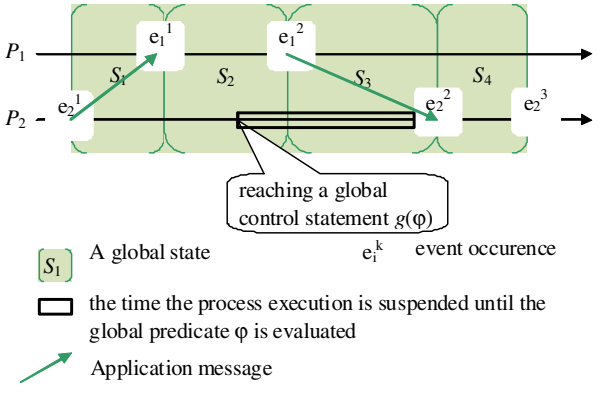


Fig. 1. Evaluation of a global predicate for synchronous global control statements in the current state mode

The behavior of parallel program controlled with synchronous reactions is deterministic. It is possible to perform a static or trace-based analysis of possible execution flows. It may be possible to obtain programs correct by construction – a predicate usually used as an invariant in proving of program correctness, can be used directly to control the execution, thus making it correct. Synchronous reactions are a natural extension to classic single-process-level control statements and therefore are easy to comprehend and employ. No special system functionalities are needed to implement them. A number of synchronous reactions variants have been described above: it requires further investigations to understand well which variant is best suited in what situations.

3.2 Asynchronous Global Control Statements

A monitor can let a process know about a fulfillment of a global predicate in an asynchronous way. The process can react by suspending current computations

and triggering a handling procedure, associated with a predicate. We designed a detailed mechanism supporting this novel approach. Because the program never waits for synchronization condition – it is interrupted when necessary – this method can lead to better performance [10]. A similar approach – asynchronous updates of data used in computations – has been taken by other researches as well [11].

Similarly to synchronous reactions, program implementation of asynchronous reactions can be decomposed into a number of components:

1. a way of expressing process states e.g. values of chosen variables,
2. global predicates definitions, created at a monitor to reflect the desired synchronization pattern,
3. modality of global predicates, defining on what global states the predicates are evaluated, e.g. SCGS as described in the previous section,
4. asynchronous notification about predicate fulfillment
5. rules of accepting the notifications (when to accept, when to ignore, when to delay a reaction)
6. code of processes separated into the main computational code and notification handling procedures

The reaction to predicate fulfillment takes place immediately when a special message from the monitor arrives. In Fig. 2(top) the arc arrows show how the application control system works in this case. An arrow begins at a global state in which a global predicate was fulfilled. As a result, the notification about the fulfillment was sent from a monitor (not shown in the figure) to a process. The heads of zigzag arrows point to the process reaction triggered immediately at the notification arrival. It is clear that the reactions can be delayed, like reaction e_2^2 at process P_2. We discuss this problem later.

There is a number of options concerning point 5 in the above list of components. We describe here static reaction scope option, according to which a syntactically defined region of code is made sensitive to predicate fulfillments. Whenever a notification about predicate fulfillment is received while the execution point is located within a notification sensitive region in the program code, the normal execution is suspended and an associated handling procedure is activated. Fig. 2 (bottom) illustrates how notifications can be accepted or discarded using static reaction scope option. A global predicate was fulfilled in global state S_1. Notifications were sent to P_2 and P_3. At P_3 the notification arrived when the process execution point was within a region sensitive to notifications, while at P_2 the notification was discarded (a crossed dashed arrow), because the process execution has left a region sensitive to notifications already (event e_2^1 stands for this leaving).

4 Global Control Structuring in Programs

We had considered asynchronous reactions as particularly interesting. Therefore we have chosen them for implementation. In [12] we describe the created parallel

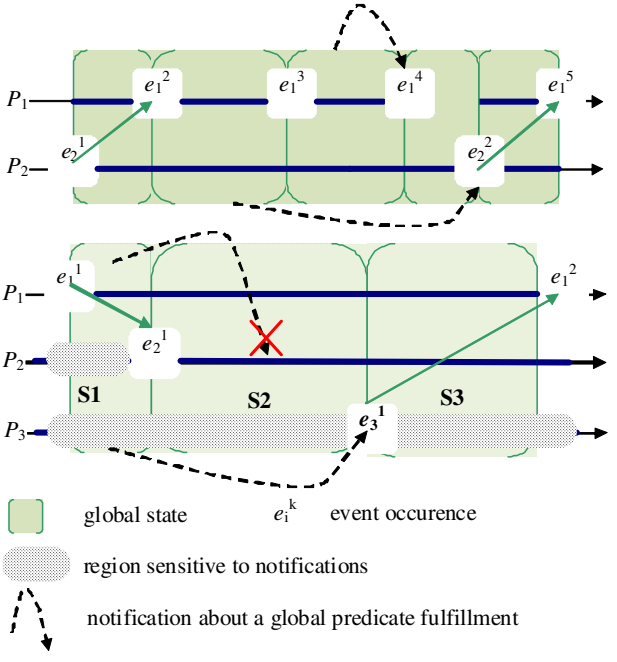


Fig. 2. Asynchronous delivery of notification about predicate fulfillment (top) and Accepting and discarding predicate fulfillment notifications (bottom)

programming system supporting process asynchronous reactions for predicate fulfillment. Now we would like to verify the other variants of reactions for global predicate satisfaction and to develop a textual language (in opposition to the graphical design method used in [12]) capable of expressing program execution flow based on global predicates. Below we present the current draft versions of the language. Our aim is to implement at least two variants of it. The implementation of asynchronous reactions, also technically challenging, should be relatively easy for us, due to our previous work. However, a proper (pre)compiler must be developed, once a final version of the control language is prepared.

To express asynchronous reactions in the code of a program we need to introduce a novel notation. The overall idea of information flow is presented in Fig. 3 on the left side, while the notation is presented below:

Global signal declaration, stating what data is contained in it:

```
Signal int sig
Computational process Pi:
Process() {
 observed v //list of state variables
 received sig // list of signal names
 watching-signal sig { //start of signal-sensitive region
 onSignal (int p){
```

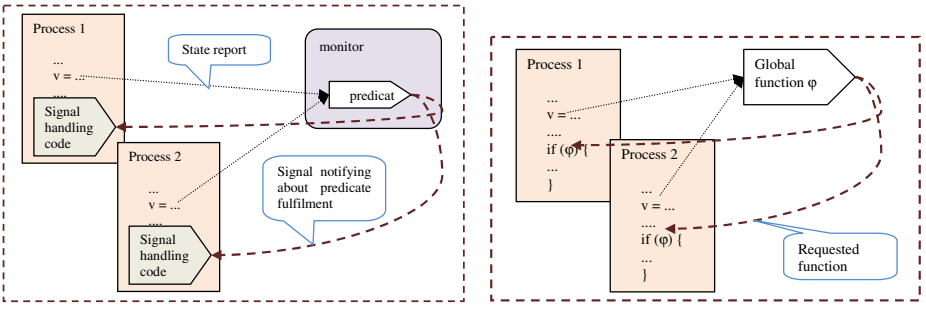


Fig. 3. Program control with asynchronous reactions (left) and Program control with synchronous global control statements (right)

```
 ...// signal handling code, possibly using the data item passed
with the signal
 }
 ...// program code
 v := newVal // new value of v is communicated to the monitor
} }
```

A monitor observing the defined global predicates can be defined as follows:

```
Monitor Mon {
 predicate pname
 observing v //list of variables declared as observed, watched by
predicate
 sent-signals sig //list of signals sent
 ... // predicate code
 ... // contains: send-signal(sig(n), proces1);
}
```

A process is connected to a proper monitor automatically during compilation by matching the lists of watched/observed variables and the lists of sent/watched signals. A value of a observed variable v from each process connected to a monitor is stored in an array. E.g. v[3] represents the recently reported value of v from the third process among processes connected to the monitor. A signal can contain some data, which can be used be the receiving process, the sample code given above illustrates this.

Synchronous global control statements can be expressed with a more standard notation. A global predicate can be used in the program code as a function and the values provided by the function can be tested. A predicate is a boolean function. However, it is much more useful to treat global predicates as functions having any declared data type. A process can use the value provided by the function in its code similarly as a value carried in a control signal is used in the case of asynchronous reactions. Therefore it is more appropriate to use the term "global functions" rather than "global predicates". Fig. 3,right-hand side, presents the general idea of using global functions to control process execution. Global function declaration:

```
  Global int p(int process) { //the parameter identifies the
process requesting function evaluation
  observing v;
  ... // code
  return r }
```

Computational process Pi:

```
Process() {
  observed v //list of state variables
  global p();
  ...
  int i = p();
  if (i>0) { ... }
  ...
  while (p() > 0) {
  ...
  v := newVal // new value of v is communicated to predicate p
} }
```

In the notation shown above, processes are associated with global functions during compilations through matching "observed" variable list with "observing" list and matching names of declared global functions with names on the "global" list in processes. The notation does not define how global functions are implemented:

- are they implemented in a monitor process, similarly as for asynchronous reactions, or
- are they computed locally by each process.

Both the modality and the choice of a particular global state, on which a global function is evaluated (see p. 3.1) can be decided freely without (almost) any change to the notation. Surely, the choices should be made in relation to each other to provide a sound functionality and efficient implementation. For "phased execution" global state selection mode an additional statement can be introduced, defining the start of the next phase. Alternatively, an implementation can assume, that a phase lasts until an observed variable gets a new value, leaving the proposed notation completely unchanged.

Program control using predicates determined on global aplication states has been verified practically by simulations and a real-world implementation [12,10]. So far we have tested the performance of a few numerical applications using the proposed approach, including parallel adaptive integration and branch-and-bound search. The obtained results and experience show, that availability of global information makes design and implementation of distributed programs easier, while the performance can be even better than using the classic - explicit message passing - approach.

5 Conclusions

Discussion of execution control implementation based on global application states monitoring was presented in the paper. Synchronous and asynchronous reactions

to the fulfilment of control predicates defined on global states enable rich functional expressiveness in control statements. The proposed diversified semantics of program execution control can be easily encoded using alphanumerical notation in de-coupled control statements of a programming language oriented towards programming of the global state-driven execution control.

References

1. Chandy, K., Lamport, L.: Distributed snapshots - determining global states of distributed systems. ACM Transactions on Computer Systems 3(1), 63–75 (1985)
2. Garg, V.K.: Methods for observing global properties in distributed systems. IEEE Concurrency 5(4), 69–77 (1997)
3. Cooper, R., Marzullo, K.: Consistent detection of global predicates. In: Proceedings ACM/ONR Workshop on Parallel Distributed Debugging, pp. 163–173 (1991)
4. Tudruj, M.: Fine-grained global control constructs for parallel programming environments. In: Bakkers, A. (ed.) Parallel Programming and Java: WoTUG-20. IOS, Amsterdam (1997)
5. Borkowski, J.: Global predicates for on-line control of distributed applications. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2004. LNCS, vol. 3019, pp. 269–277. Springer, Heidelberg (2004)
6. Borkowski, J.: Parallel program control based on hierarchically detected consistent global states. In: International Conference on Parallel Computing in Electrical Engineering (PARELEC 2004), pp. 328–333. IEEE, Los Alamitos (2004)
7. Babaoglu, O., Marzullo, K.: Consistent global states of distributed systems: fundamental concepts and mechanisms. In: Distributed Systems. Addison-Wesley, Reading (1995)
8. Stoller, S.D.: Detecting global predicates in distributed systems with clocks. Distributed Computing 13(2), 85–98 (2000)
9. Skillicorn, D.B., Hill, J.M.D., McColl, W.F.: Questions and answers about BSP. Scientific Programming 6(3), 249–274 (1997)
10. Borkowski, J., Tudruj, M., Kopanski, D.: Global predicate monitoring applied for control of parallel irregular computations. In: 15th Euromicro Conference on Parallel, Distributed and Network-Based Processing PDP 2007, Naples, Italy. IEEE, Los Alamitos (2007)
11. Bahi, J., Contassot-Vivier, S., Couturier, R.: Dynamic load balancing and efficient load estimators for asynchronous iterative algorithms. Transactions on Parallel and Distributed Systems 16(4), 289–299 (2005)
12. Tudruj, M., Borkowski, J., Kopanski, D.: Graphical design of parallel programs with control based on global applications states using an extended p-grade system. Distributed and Parallel Systems 777, 113–120 (2004)

Distributed MIND – A New Processing Model Based on Mobile Interactive Documents

Magdalena Godlewska and Bogdan Wiszniewski

Gdańsk University of Technology
Faculty of Electronics, Telecommunications and Informatics
ul. Narutowicza 11/12, 80-233 Gdańsk
{magdal,bowisz}@eti.pg.gda.pl

Abstract. Collaborative computing involves human actors and artificial agents interacting in a distributed system to resolve a global problem, often formed dynamically during the computation process. Owing to the open nature of the system and non-cooperative settings, its computations are in general non-algorithmic, i.e. their outcome cannot be calculated in advance by any closed distributed system. Authors advocate for a new processing model, based on exchange of documents implemented as autonomous and mobile agents, providing adaptive and self-aware content as interface units.

Keywords: collaborative computing, intelligent documents.

1 A Mobile Document Concept

Collaborative computing processes are often based on knowledge that shall be created and acquired dynamically. It involves human actors who cooperate within a certain organizational structure by creating and exchanging documents. Their cooperation is not algorithmic, thus process outcome cannot be calculated in advance by a computer system. Moreover, documents being exchanged are the only visible interfaces to the respective processes, and constitute independent units of information necessary to elaborate a final result upon which all related documents have to be archived in a form enabling their future reuse.

A key feature of this scheme is an exchange of information objects, which are electronic documents, and owing to their logical structure mark-up, are readible simultaneously to humans and computers. Collaborative computing systems of today are typically based on a pool of servers dealing with electronic documents as passive information units, rather then autonomous and interactive objects. It may be expected that augmenting electronic documents with embedded services (functionality) will enable realistic integration of human agents and their programmatic counterparts in one computation process.

MIND builds upon two concepts (see Figure 1): an automatic XML data binding, which allows for representing units of information in an XML document as a set of functional objects in computer memory, and mobile agents augmenting these objects to make them capable of migrating from one computer to another

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 244–249, 2010.

to continue execution. Owing to that, a static document template represented by a schema is converted to component objects, which as autonomous agents will next implement their individual actions at designated destinations. This process is initiated by a *document originator*, who designs a *hub document* based on a schema defining a logical structure and functionality of a document template of interest.

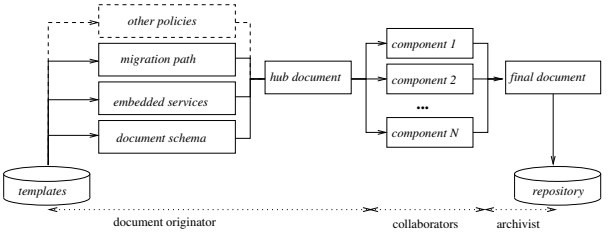


Fig. 1. Lifecycle of a MIND document

Key parts of the hub document are: *components* with information (data) content, which upon unmarshalling become mobile objects, document *migration path* (specified in a form of a document workflow) and specification of *embedded services*. Embedded services are implemented with plugins, expanding dynamically document functionality, according to the current context of the local host upon arrival of the agent. Users who interact with components incoming to their local workstations are *collaborators*; they make decisions based on the content of other components they have access to, and if necessary create new components in the form of dynamic annotations, linked to other components of interest. Upon reaching its goal, each migrating object (document component) returns to its document originator to be marshalled with other returning objects back to an XML document. Files with XML documents are archived by an archivist in a repository for future use (mainly extration of accumulated knowledge).

2 Technological Challenges for MIND

Implementation of the MIND concept involves specific combination of just a few existing Web technologies. However, their survavibility as standards is important to consider the MIND architecture as a reasonable alternative to data-centric computation models.

Below a quick review of the MIND core technologies and their state-of-the-art has been given to assess prospects of a new distributed processing model advocated in the paper.

2.1 Logical Document Structure Definition

With the advent of XML, a uniform mark-up notation for describing logical structure of data has been established as a global standard. Representing the

logical structure of documents and their components is important for MIND, since according to the concept illustrated in Figure 1 documents are converted to interactive components, enabling collaborators to manipulate their content. Several languages have been proposed to model a schema, to which a structure of a particular XML document should conform in a checking process called validation. Any schema language is a certain compromise between its power of expression, simplicity, and features, being more or less suitable for particular types of applications. The problem is to choose a schema language best fitted for the class of documents supporting collaborative computing processes today and in the future. Equally important is the support by organizations and vendors providing standards and tools to promise survival of the chosen language and guarantee forward compatibility of archived documents. For example, Document Type Definition (DTD), historically rooted in the SGML mark-up language, provides a very simple grammar mechanism and allows for structure rules to be parameterized and customized. It has also an excellent vendor support and prospects for survival as a default schema language. Unfortunately, its expressiveness is low, as DTD has been shown to be in the most restrictive local tree grammar (LTG) subclass of regular tree grammar (RTD) languages [7]. The intent of XML Schema language is to reconstruct the facilities provided by DTDs parameter entities and marked sections into a full type-lattice system with type inheritance, type extensions and type restriction. Although relatively more complex to use than DTD, it has been shown to be a member of a less restrictive single-type tree grammar (STG) subclass [7]. Algorithms for constructing a unique interpretation of a document when validating it against a schema written in any STG language are widely used. Finally, RELAX NG [2], has been so far the least restrictive schema language, as it properly belongs to the topmost class of RTG languages. This freedom, however, implies difficulties in checking for ambiguity in matching regular expression patterns (an algorithms for that has been discovered later on [4]).

2.2 Document Component Data Binding to Functional Objects

In order to convert an XML document into a set of objects enabling manipulation of data one needs a tool for XML data binding to these objects. A potentially serious challenge may be that XML data binding tools cannot operate on document fragments, i.e., cannot extract data from one or more fragments of an XML document, expose that data using schema-specific objects, and re-write those fragments to the document, leaving the rest of the document unchanged. This is a real problem in workflow scenarios, where many applications work on one document, each modifying and passing part of it to the next application. Architecture of MIND outlined in Figure 1 can cope with that challenge, since document fragments extracted once from the global hub document are bound to the application run by a document originator to objects with embedded functionality driven by externally defined policies and making these objects active "applications", capable of operating on their related document fragments.

2.3 Augmenting Document Component Objects with Mobility

In order to enable migration of document components to remote sites for processing, they have to be converted to mobile objects (agents). Since their invention over a decade ago and dozens of different platforms developed, agents have been claimed to become a breakthrough in distributed computing with such obvious benefits as loose coupling, adaptability and support of heterogeneous systems. Surprisingly, it is relatively difficult to point to successful large-scale implementations of agent systems. Probably the reason is that the basic paradigm of agent technology involves functional specialization and autonomous interaction with the local environment, narrowing the class of realistic applications to just a few, such as resource management and maintenance of complex systems, or delivery of personalized content and e-commerce, for each of which a volume of agents should rather not be excessive. MIND concept of representing documents as functionally autonomous and mobile objects requires, however, taking a closer look at *scalability* of the most popular agent programming platforms in terms of excessive message sending and agent migration actions, since the volume of component objects might be sometimes as large as few thousands for particularly complex collaborative computing processes (eg. court trials). A recent survey of a variety of agent platforms representing the current state in the field [6] indicates that for agent applications of a moderate size (up to 100 agents, each one performing few hundred movements and communications) the differences in reliability and performance are nor significant, and only some platforms have problems with reliable execution when agents move and communicate; agents that only move usually have less problems. Another experiment [3] aimed at flooding of a single node with incoming agents indicates that a JADE based application could reliably perform with up to 800-1000 incoming agents. These results are promising for MIND applications. Another issue is *stability* of the selected agent platform to guarantee survivability of MIND technology. An important point of reference here are the abstract platform and agent management specifications provided by FIPA; so far JADE, offers a platform with the strongest resemblance to this specification.

2.4 Workflow of a Document Component Object

Collaborators interact in a virtual space with document component objects to co-edit or otherwise modify their content. Inspiration for that form of collaboration has been development of open source software by large groups of programmers. A key feature of that is a global scope of work, hard to achieve with paper-based documents. Each collaborator processes the assigned piece of content, by executing elementary operations like insert, cut and replace, while a system supporting the society of workers has to assure consistency and completeness of all so produced components. Historically, there have been two possible approaches for such a collaboration: *pessimistic*, where a unique copy of the edited document is shared at a central server, and *optimistic*, where before edition, a document is replicated and copies are sent over a network to each respective co-worker, who edits locally the received copy. Disadvantages of the former are the necessity

to stay on-line by all collaborators and no provision for concurrency, while for the latter a need to implement a complex mechanism for transforming editing actions performed locally with messages received from remote sites. Unfortunately, many algorithms already published in the literature have been found flawed in that regard as not assuring consistency of document copies processed in parallel [1].

MIND proposes a third, *realistic* approach, with a document transformed into a set of objects forming a certain hierarchy implied by its logical structure, and having their own local memory and methods. One advantage is the potential of using well-defined object-oriented mechanisms for document processing, in an open multi-agent system fashion, allowing for dynamic expansion and migration of document components.

The content of distributed document components, augmented with mobility, enables information flows that can be organized into work processes involving intellectual resources of collaborators. These processes can be formally described with a *workflow definition language* as patterns of activities, which together with embedded services of document components, implemented with plug-in actions, and predefined migration paths, implicitly related to the document component type and semantics, constitute a sort of an actionable meaning of the document content. Owinf to XSLT, forward compatibility of MIND may be secured practically for any workflow definition language based on XML.

2.5 Location Control of a Document Component Object

During processing along their respective workflows, components of a distributed document may need to refer to one another in the form of links, citations or bookmarks. The common addressing structure for the Web, based on the URLs, combining logical names with physical IP addresses to identify and locate digital objects, fails whenever the resources are moved between locations – what is the case of mobile MIND objects. Moreover, some of these mobile objects may be dynamic in the sense that they are created after originating the workflows of document components defined by a schema of the document template. One example are *dynamic annotations* that may be created and linked by knowledge workers to other components at any time during the document lifecycle (see Figure 1). The notion of a persistent identifier is needed, i.e. the one that tracks specific objects it refers to regardless of their physical location. Two primary persistent identifier applications have emerged and are strongly supported: the Persistent URL (PURL) [5], and the Handle System [8]. Both approaches provide registration and resolution services to map the persistent identifier to the current physical location of the digital object of interest and implement a sort of a redirection mechanism. PURL assigned to a mobile document points to the special resolver record in a resolver table maintained by a dedicated PURL server; the resolver record contains information to redirect the PURL to the current URL of the object, while the resolver table is updated each time any actual URL stored in it changes. Owing to this, the referred objects PURL does not change when the object migrates. The Handle System is an interoperable

network of distributed resolver servers, linked through a Global Handle Server; a local Handle Server can resolve any handle through the global server to the current URL of the migrating object. Besides that the resolution of one handle to multiple objects, to other handles or even email accounts is possible. Persistent identification of MIND objects can adopt any of these mechanisms for dynamic annotations management.

3 Conclusions

Document-centric processing model advocated in the paper provides a natural mechanism for incorporating intelligence of human actors in computations of an open multi-agent system. Brief review of candidate Web technologies presented in the paper indicate that they can widely support MIND, despite of some minor concerns regarding forward compatibility of the model, implied by their survivability as standards. In that regard adoption of XML based notations is certainly a must, so the first prototype implementation of MIND developed by the authors involved: XML Schema for logical document structure definition, XML Beans for document data binding and XPDL for component workflow specification. The agent platform chosen for this implementation was JADE, which seems to be stable as well. Early experiments with the MIND prototype indicate that intelligent, autonomous and adaptive document content enabled by MIND should significantly speed up and ease complex interaction and leverage knowledge bound organizations with virtual collaboration to enable resolving non-algorithmic decision problems, such as court trials, integrative bargaining, medical consultations, or crash investigations.

References

1. Oster, G., Molli, P., Urso, P., Imine, A.: Tombstone transformation functions for ensuring consistency in collaborative editing systems. In: Proc. Collaborative Computing: Networking, Applications and Worksharing (2006)
2. Clark, J., Murata, M.: RELAX NG specification. OASIS TC Specification (December 3, 2001), http://www.oasis-open.org/committees/relax-ng
3. Chmiel, K., Gawinecki, M., Kaczmarek, P., Szymczak, M., Paprzycki, M.: Efficiency of (JADE) agent platform. Scientific Programming 13 (2005)
4. Murata, M., Lee, D., Mani, M.: Taxonomy of XML schema languages using formal language theory. In: Proc. Extreme Markup Languages (2000)
5. PURLS. A project of OCLC research. OCLC Online Computer Library Center, http://www.purl.org
6. Trillo, R., Ilarri, S., Mena, E.: Comparison and performance evaluation of mobile agent platforms. In: Proc. 3rd Int. Conf. on Autonomic and Autonomous Systems, ICAS 2007 (2007)
7. Gao, S., Sperberg-McQueen, C.M., Thompson, H.S. (eds.): W3CXML Schema Definition Language (XSD) 1.1 Part 1: Structures, Part 2: Datatypes. W3C Working Draft, January 30 (2009), www.w3.org/TR/xmlschema11-1/
8. Handle System. Unique persistent identifiers for internet resources, http://www.handle.net

A Framework for Observing Dynamics of Agent-Based Computations

Jarosław Kawecki and Maciej Smołka

Institute of Computer Science, Jagiellonian University, Kraków, Poland
smolka@ii.uj.edu.pl
http://www.ii.uj.edu.pl/~smolka/

Abstract. The paper contains a description of a framework designed for observing dynamics of mobile-agent computational applications. Such observations are thought to provide a basis for the experimental verification of an existing stochastic model of agent-oriented distributed computations. Some test results are also provided, which show that the proposed observational environment does not disturb an observed system's dynamics.

1 Introduction

Multi-agent systems (MAS) are considered one of significant paradigms for distributed system design in the industry (cf. [1]). In the science they are used to solve some complex problems such as evolutionary global optimization (cf. [2], [3]). However it is still not very common to apply multi-agent paradigm in the implementation of large-scale distributed computational systems, even if the idea of self-organizing computational application being a collection of mobile tasks which migrate over a network according to a diffusion-based policy in order to find the best environment for computations is known for several years ([4]). Thus it was quite straightforward to merge the two ideas, i.e. to enclose a computational task together with its data in a mobile agent box, giving the agent the abilities to migrate, to communicate with other agents, to split itself and to sense its environment properties, and finally providing the agent with some logic to decide which abilities to use in order to perform its task. Such a computational multi-agent system has been constructed ([5], [6]) on the basis of Java/CORBA platform. Sec. 2 describes its main features.

During the development of the system many theoretical questions has been raised. As an answer a formal model of multi-agent computations has been proposed ([7], [8]). It describes a multi-agent computational application as a controlled Markov chain. The model provides us with the definition of optimal scheduling and some results on the existence and characterization of optimal scheduling strategies. The model is briefly described in Sec. 3.

However the model itself needs experimental validation. The first step towards this goal is the construction of a framework for monitoring the dynamics of quantities appearing in the model. The present paper describes such a framework. It is designed as an extensible additional module for the above-mentioned

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 250–259, 2010.

computational MAS platform. Design and implementation issues are covered in Sec. 4, some test results showing good features of the monitor are contained in Sec. 5.

2 Computational MAS

As the above-mentioned computational MAS is the basis for our new-projected framework let us first briefly describe its main architectural principles. For a more complete description and some implementation details we refer the reader to [5] and [6].

The architecture of the system is composed of *a computational environment* (MAS platform) and *a computing application* being a collection of mobile agents called *Smart Solid Agents* (SSA). The computational environment is the triple $(\mathbf{N}, B_H, perf)$, where:

$\mathbf{N} = \{P_1, \ldots, P_N\}$, where P_i is a MAS server called *a Virtual Computational Node* (VCN). Each VCN can maintain a number of agents.

B_H is the connection topology $B_H = \{\mathcal{N}_1, \ldots, \mathcal{N}_N\}, \mathcal{N}_i \subset \mathbf{N}$ is a direct neighbourhood of P_i (including P_i as well).

$perf = \{perf_1, \ldots, perf_N\}, perf_i : \mathbb{R}_+ \to \mathbb{R}_+$ is a family of functions where $perf_i$ describes the relative performance of VCN P_i with respect to the total memory request M^i_{total} of all agents allocated at the node.

The MAS platform is responsible for maintaining the basic functionalities of the computing agents. Namely it delivers the information about the local load concentration, performs agent destruction, hibernation, partitioning and migration between neighbouring VCN's and finally supports the transparent communication among agents.

We shall denote an SSA by A_i where index i stands for an unambiguous agent identifier. Each A_i contains its computational task and all data necessary for its computations. Every agent is also equipped with a shell which contains the agent logic. At any time A_i is able to denominate the pair (E_i, M_i) where E_i is the estimated remaining computation time measured in units common for all agents of an application and M_i is the agent's RAM requirement in bytes. An agent may undertake autonomously one of the following actions: *continue* executing its internal task, *migrate* to a neighbouring VCN or decide to be *partitioned*, which results in creating two child agents.

3 Formal Model Overview

In this section we introduce some key concepts appearing in our formal model of computing multi-agent systems. The detailed description of the model can be found in [7] and [8].

The main idea behind the model is the following. Instead of considering a single agent's behaviour we observe the time evolution of some global quantities characterizing the state of the whole computational application. The set

of state variables may typically include: the total number of allocated agents with respect to a VCN, the total remaining time of computations with respect to a VCN, the total memory requirement of agents with respect to a VCN. In addition some control variables have been introduced in the model. Namely we control the number of agents migrating between nodes, the number of agents splitting themselves and optionally the number of agents being hibernated or dehibernated. Finally equations of system evolution have been formulated describing a computational multi-agent system as a controlled Markov chain. It allows us to answer many fundamental questions (such as the question about the existence of optimal control strategies) by means of the stochastic control theory machinery.

Such a model needs an experimental verification. We are going to observe the dynamics of existing and new computational applications in order to check if the model describes them properly. But to this end we need a monitoring infrastructure which would register the time evolution of interesting application parameters and which, on the other hand, would not disturb the system behaviour significantly.

4 Monitoring Subsystem Architecture

4.1 Overview

The subsystem monitoring the agent dynamics is designed as a wrapper for a single MAS server and it focuses only on tracing the state of its host. Different wrappers do not communicate with each other hence they do not generate any network traffic. However such an approach needs an external synchronization mechanism to gather data with global time stamps for further processing and analyzing.

The monitor uses the SNTP protocol to set global time among servers. Each of monitors carries its own time service. Measurements taken during a monitor's work are stamped with time provided by this service. To stay accurate the time service periodically sends synchronization requests to the NTP server. The result is stored as a time offset and it is used to calculate the global time.

Calculations on the MAS platform are performed by agents. Significant aspects of the agent life-cycle activities (such as agent creation, migration or destruction) are traced using an event-driven approach. Namely every time an agent is born, leaves a server, enters another server or dies a trigger can be fired. It is marked with the current time stamp and with the agent identifier, so the whole agent traffic can be reconstructed after the experiment. The event-driven approach is reliable because it takes into account every event that occurs at the monitored server. However it is not possible to implement triggers totally outside the platform environment. The event infrastructure needs code integration with the platform server and the agent.

Some quantities should not be traced using the event-driven approach due to performance issues. Triggers are not efficient when a quantity value changes really often. On the other hand some quantities can change in an unpredictable

way, so there is no place where the trigger could be hooked. A snapshot mechanism is meant to support triggers in such situations. It is a kind of task that periodically takes a snapshot of a server's parameters. Such an approach avoids drawbacks of the event-driven approach and allows to gather more data for further analysis. Without it the server memory consumption or the processor utilization could not be traced.

4.2 Implementation Issues

A single monitor server consists of three main parts that are sequentially initialized (see Fig. 1). The first to launch is the monitor's observation subject, i.e. a MAS server (cf. [5]). The server is running on a dedicated thread. The time service is started next. It is used for periodical sending of SNTP requests to the time server in order to keep the most up-to-date offset to the global time. Third essential component called the Experiment is started as the last. It initializes triggers, snapshots and file output where measurement data are stored. The global time provided by the time service is used to stamp measurements. Experiment class is closely coupled to the MAS server and this dependency is bidirectional. Using snapshots the Experiment observes the state of the server, while the server fires triggers which are stored in the Experiment.

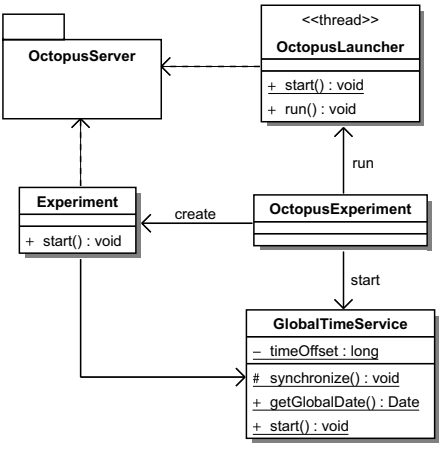


Fig. 1. Main components of a server monitor (UML class diagram)

The time service uses the SNTP client to get the offset value from the time server. This is performed periodically. To this end the service spawns its own thread which sleeps during the most of its lifetime. It is only awaken once for a defined period and then it tells the service to update the time offset. Each time the update is done the service invokes the synchronization trigger. It allows us to store the retrieved time offset in an output file.

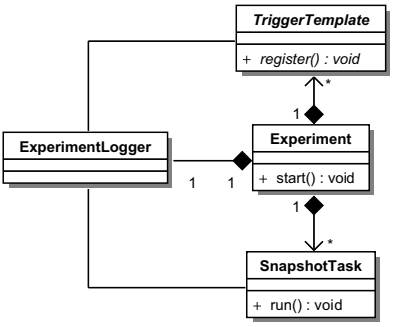


Fig. 2. Experiment (UML class diagram)

The main responsibility of the Experiment is to create all configured snapshots, triggers and instantiate the experiment logger (see Fig. 2). When all these items are created the experiment may be started. Then the experiment logger is initialized, all snapshots are scheduled and finally all triggers are registered. It means that since then triggers are able to receive calls from the MAS code. It is worth noticing that scheduling snapshots does not mean an immediate execution. The idea of the snapshot is to catch the consistent state of the whole platform. To this end two conditions must be met: all snapshots start at the same time and they are repeated with the same frequency. Both of these parameters are passed by the configuration file. The start time is defined as the globally defined moment common for all monitors. Before that moment no snapshots are taken, therefore no experiment activity should be performed. Only after the start time the platform is capable to work and all parameters are properly monitored.

A snapshot itself (see Fig. 3) does not observe a server state directly. It only provides environment for execution of measurements such as memory or processor utilization. The static part of the snapshot allows to assign some

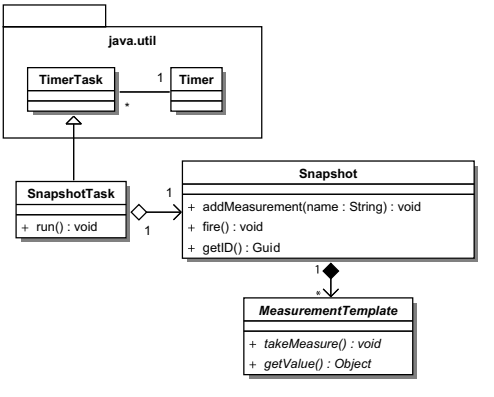


Fig. 3. Snapshots (UML class diagram)

measurements. Then the dynamic part runs the snapshot. As s consequence all the assigned measurements are taken and sent to the experiment logger. As mentioned before the snapshot runs periodically. The facility that provides this behavior belongs to the Java standard library. The Timer object can schedule running objects whose classes extend the TimerTask abstract class. There is a major advantage of using the Java Timer, namely it is able to correct delays in the task execution. As a result the snapshots do not accumulate time shift.

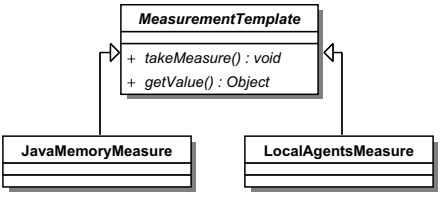


Fig. 4. Measurements (UML class diagram)

A measurement (Fig. 4) is a piece of code which is responsible for measuring one parameter of the server. It takes the measure, stores the result and makes the value of the measure available for the experiment logger. The measurement is always considered in the context of a snapshot. Currently two kinds of measurements are implemented. The memory measurement monitors free memory. The agents measurement checks how many agents are currently located on a MAS server.

A trigger (Fig. 5) from a point of view is similar to a measurement described before. It is also connected with a single parameter of the MAS server. Despite

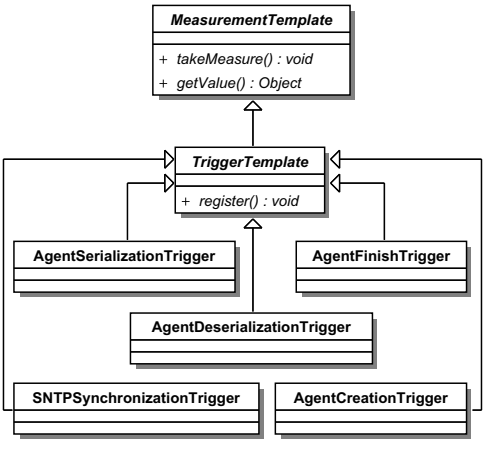


Fig. 5. Triggers (UML class diagram)

the similarity the trigger can work without any snapshot. Not being a part of a snapshot has a downside effect. The code of trigger needs to be executed directly from the MAS platform code. Two MAS platform classes had to be modified in order to launch triggers related to agents: SerializedObjectStreamService and TaskBase (cf. [5]). Code changes were kept as small as possible. Currently five types of triggers are implemented. Serialization and deserialization triggers are responsible for tracing migration and hibernation of agents. Next two trigger types are designed for monitoring agent creation and destruction, hence in particular they can be used to observe partitioning. The last trigger type is fired when a synchronization with the time server occurs.

5 Monitor Performance Tests

The MAS platform monitor was designed to impose the lowest overhead possible. Of course since the monitor integrates with the platform code, it impacts utilization of such resources as processor, memory and network. Hence in order to determine the scale of the actual overhead some tests have been performed. They compared the total execution time for the *Subdomain-by-Subdomain* (SBS) distributed linear solver (cf. [5]).

The platform consisted of 12 PCs organized in a star topology and equipped with the Fedora operating system. There was also one PC outside the platform dedicated for the administration purposes such as: launching MAS servers in a specified order, initializing and deploying SBS computational agents and obtaining the test results. The monitor configuration consisted of:

- SNTP synchronization run every 10 seconds using `vega.cbk.poznan.pl` NTP stratum 1 server;
- all 5 triggers enabled (i.e. synchronization, agent serialization, deserialization, creation and destruction);
- first snapshot run every second tracing current number of agents located on a server;
- second snapshot run every 2 seconds logging free server memory.

All agents were deployed on the central server (s213-03) and then they were allowed to migrate according to a diffusion-based policy (cf. [6]).

The tests were organized into 16 groups. They varied by the number of agents involved in the computations and the size of the matrix computed by one agent. The tests took into account the total computation time. Each of the test groups was run 8 times for the unobserved MAS platform and other 8 times for the platform observed by the monitor.

Figs. 6 and 7 show that computations on the platform with the monitor are not significantly slower (at least in average) than those on the platform without the monitor. Thus the test results allow us to treat observations performed by the monitor as quite reliable, which means that the monitor does not disturb the agent dynamics.

Size of Matrix	Number of Agents: 32	64	128	256
72	54875ms	106125ms	194500ms	362250ms
144	66250ms	105375ms	206250ms	391750ms
288	67875ms	118125ms	232625ms	448500ms
576	108375ms	191500ms	348750ms	677857ms

Size of Matrix	Number of Agents: 32	64	128	256
72	54375ms	103375ms	196875ms	353125ms
144	62125ms	105125ms	203125ms	395375ms
288	66375ms	117375ms	231625ms	429625ms
576	106500ms	187500ms	348500ms	678000ms

Fig. 6. Average execution time without (on the left) and with (on the right) the monitor

Size of Matrix	Number of Agents: 32	64	128	256
72	3257ms	8462ms	11576ms	18012ms
144	27954ms	7761ms	9189ms	20553ms
288	6030ms	5372ms	4998ms	29270ms
576	3672ms	9539ms	7462ms	21557ms

Size of Matrix	Number of Agents: 32	64	128	256
72	2288ms	7936ms	12752ms	2368ms
144	10959ms	9545ms	10105ms	28026ms
288	2118ms	4973ms	10086ms	15588ms
576	2449ms	4243ms	7665ms	15344ms

Fig. 7. Execution time standard deviation without and with the monitor

6 Sample Observation of Formal Model Quantities

Next let us present some capabilities of the monitor in the area of observing quantities appearing in the previously-described formal model. The data presented in this section were obtained using the same topology as for the verification purposes. 256 agents were created at the external server and deployed onto the

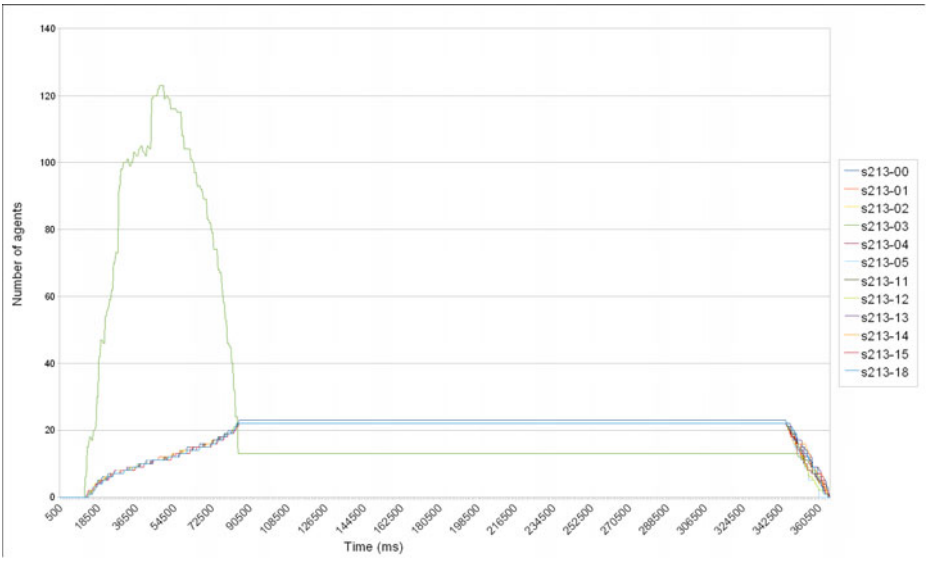


Fig. 8. Number of agents on platform servers

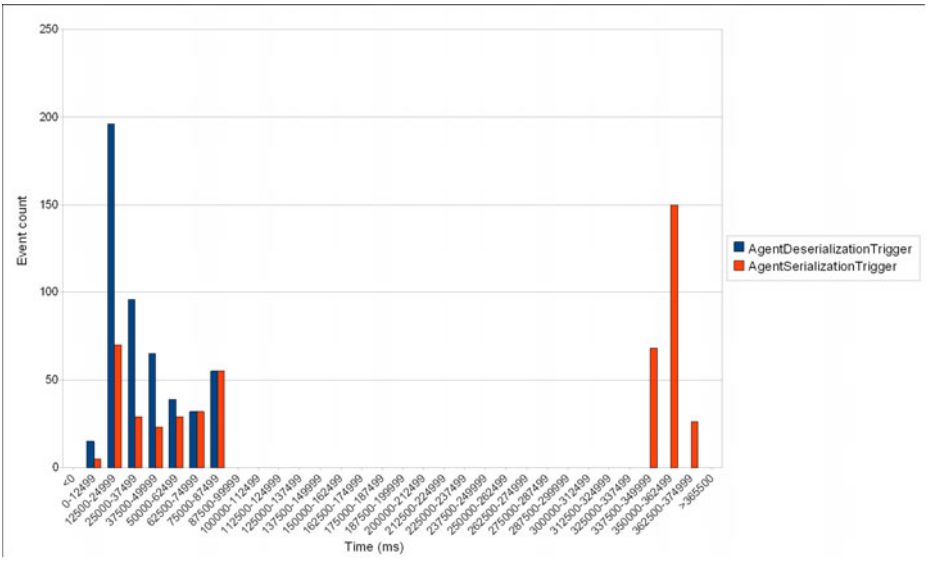


Fig. 9. Serialization and deserialization events

platform. Each of them resolved a linear equation system with the matrix of size 72. The data is presented with the precision of 500 milliseconds.

Fig. 8 shows the number of agents allocated on platform servers. The data were gathered using snapshots. Another view of the same quantity is given by Fig. 9 which shows serialization and deserialization activity on the servers registered by triggers. It turned out that SBS was quite predictable when it comes to the migration of agents. The first phase lasted about 90 seconds. Agents were deployed to the s213-03 server which was the centre of the star connection topology. There was noticeable increase of agent number at this server till 50th second. Then the immigration to the s213-03 stopped and agents were only migrating to leaves of the topology. After allocation agents to servers the migration process ended and all agents were busy with computations. The computation phase lasted approximately the same amount of time for all agents. At the end every agent returned the computation results to its parent and left the platform.

7 Conclusions and Further Research

An important goal for the designed monitoring subsystem was to avoid any disturbance in observed system dynamics. As test results suggest, the goal has been achieved. Moreover the constructed framework introduced only very small changes in the MAS platform code. The resulting framework is already quite useful because it can measure all basic formal model quantities, yet it is extensible enough to forecast some future needs such as new kinds of triggers or snapshot measurements possibly introduced in more sophisticated versions of the model.

A natural consequence of the monitor construction shall be gathering observation data from computational applications other than SBS and validating the model through the analysis of the data.

References

1. Wooldridge, M.: An Introduction to Multi-agent Systems. Wiley, Chichester (2002)
2. Cetnarowicz, K., Kisiel-Dorohinicki, M., Nawarecki, E.: The application of evolution process in multi-agent world (MAW) to the prediction system. In: Tokoro, M. (ed.) Proceedings of 2nd International Conference on Multi-Agent Systems (ICMAS 1996), Osaka, Japan. AAAI Press, Menlo Park (1996)
3. Byrski, A., Kisiel-Dorohinicki, M.: Agent-based evolutionary and immunological optimization. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4488, pp. 928–935. Springer, Heidelberg (2007)
4. Luque, E., Ripoll, A., Cortés, A., Margalef, T.: A distributed diffusion method for dynamic load balancing on parallel computers. In: Proceedings of EUROMICRO Workshop on Parallel and Distributed Processing, San Remo, Italy, January 1995, pp. 43–50. IEEE Computer Society Press, Los Alamitos (1995)
5. Uhruski, P., Grochowski, M., Schaefer, R.: Multi-agent computing system in a heterogeneous network. In: Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC 2002), Warsaw, Poland, September 22-25, pp. 233–238. IEEE Computer Society Press, Los Alamitos (2002)
6. Grochowski, M., Schaefer, R., Uhruski, P.: Diffusion based scheduling in the agent-oriented computing systems. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2004. LNCS, vol. 3019, pp. 97–104. Springer, Heidelberg (2004)
7. Smołka, M.: A formal model of multi-agent computations. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 351–360. Springer, Heidelberg (2008)
8. Smołka, M.: Task hibernation in a formal model of agent-oriented computing systems. In: Bubak, M., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2008, Part III. LNCS, vol. 5103, pp. 535–544. Springer, Heidelberg (2008)

HyCube: A DHT Routing System Based on a Hierarchical Hypercube Geometry

Artur Olszak

Institute of Computer Science, Warsaw University of Technology
A.Olszak@ii.pw.edu.pl

Abstract. This paper presents a DHT routing system based on a hierarchical hypercube geometry. An approach employing a novel variable metric adopting the Steinhaus transform is presented. The use of this metric and the hierarchical hypercube geometry allows to reach very good performance of the network and a very high level of resilience to node failures. The architecture of the network and the routing algorithm are presented. The results of the simulations have been included in the paper and compared with existing solutions.

Keywords: peer-to-peer network, distributed hash table, Steinhaus transform, hierarchical hypercube.

1 Introduction

Recently, we observe an increase in interest in peer-to-peer networks, in particular routing algorithms. Most of the networks currently developed are based on a distributed hash table algorithm (DHT). DHT systems store key-value pairs and the set of keys is distributed among the nodes in the network. The nodes in DHTs usually share storage or computation resources with other nodes. Each node has its unique identifier. The resources (values) are located by searching for a node responsible for the resource key. Usually this is the node (or nodes) with the *id* closest to the key. Every node stores a set of references to other nodes - a routing table. The routing table is built in a way that allows to send a message, decreasing the distance left to the destination node in each routing step (distance between identifiers, according to the chosen metric). DHT systems can be divided into groups according to their geometry (connections graph) which defines the overlay network. The overlay network implies the structure of the routing tables. The most important DHT geometries are ring (e.g. *Chord* [1]), XOR metric (e.g. *Kademlia* [2]), tree (e.g. *Pastry* [3]), hypercube (e.g. *CAN* [4]), butterfly (e.g. *Viceroy* [5]). The geometry influences mostly the route lengths, but it also affects the level of resilience to node failures.

In [6], the authors analyze the impact of the routing geometry on static resilience[1] and on the average route length and present the results of the simulations. Authors claim that flexibility in the next hop selection (which can be

[1] The ability of the network to route messages between nodes in the presence of node failures without the aid of recovery mechanisms.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 260–269, 2010.

measured by the number of nodes in the routing table to which a message can be routed) has an impact on static resilience. Different geometries imply different levels of flexibility in the next hop selection and the simulations prove that it has a great impact on the static resilience.

In some DHT systems, nodes maintain sets of so called sequential neighbors. Sequential neighbors are the preceding node and the following node (existing nodes with identifiers closest to the node's identifier that appear directly before or after it according to a certain sequence). Nodes form a logical ring using connections with predecessors and successors. The leaf set in *Pastry* and the predecessor and the successor in *Chord* are examples of sequential neighbors. Messages can always be routed to these nodes, decreasing the distance left to the destination - depending on the direction in which the destination is located, either the predecessor or the successor is chosen. In order to increase the level of resilience to node failures, in some DHT networks, nodes maintain sets of several preceding and several following nodes so that packets can be routed to the next hop, even in the presence of failures of many nodes. The results of the experiments presented in [6] show that sequential neighbors greatly improve static resilience but the average route length may significantly increase. For this reason, nodes usually store some constant number of sequential neighbors to provide tolerance to node failures, whereas efficient routing is provided by routing tables specific for particular DHT algorithms.

This paper presents the routing geometry and the routing algorithm of *HyCube* - a DHT peer-to-peer network based on a hierarchical hypercube geometry. Routing is based on a variable metric adopting the Steinhaus transform which defines distances between nodes. Using such a metric results in a very high degree of flexibility in the next hop selection. In comparison to schemes using sequential neighbors, this approach allows to achieve a very high level of static resilience and a shorter average path length in the presence of node failures.

The rest of this paper is organized as follows. Section 2 presents the design of *HyCube*, including a description of the routing geometry and the routing algorithm. Section 3 describes the metric used in *HyCube*. Experimental results are presented in Section 4. Section 5 concludes.

2 HyCube

In this section, the routing procedure of *HyCube* is described. Section 2.1 presents the routing geometry, Section 2.2 describes routing tables that nodes maintain to support the routing procedure, Section 2.3 presents the routing algorithm.

2.1 Routing Geometry

The routing geometry of *HyCube* is a combination of a tree geometry and a hypercube geometry. It is based on *Plaxton mesh*[7] but nodes are also logically located on a d-dimensional torus. Nodes in the network are distributed in vertices of a hierarchical hypercube. Figure 1 presents the structure of the network for 3 dimensions and 2 hierarchy levels.

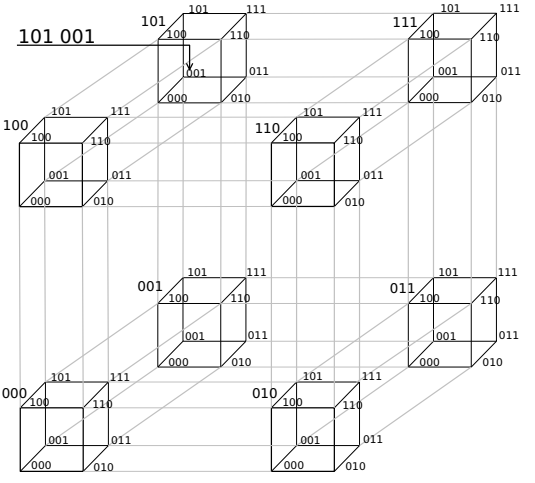


Fig. 1. A hierarchical hypercube (3 dimensions and 2 levels of hierarchy)

A hierarchical hypercube is a hypercube whose vertices are also hypercubes. Vertices of the hypercubes at the lowest level are positions which may be occupied by nodes. The position of a node in a hierarchical hypercube is determined by its identifier. The identifier of a node is a string of d-bit groups determining the positions of the node in hypercubes at particular levels (starting with the highest level). The position in a hypercube is a string of bits corresponding to the location of the node in the hypercube in particular dimensions. The length of the identifier equals $d \cdot l$, where d is the number of dimensions and l is the number of levels. The structure of a hierarchical hypercube and a tree are isomorphic. However, looking at it as on a hierarchical hypercube gives an idea of the spatial arrangement of the nodes in the network. The numbers formed from bits corresponding to particular dimensions relate to the coordinates of the node in these dimensions in the system of coordinates with the center in point 0. Thus, considering the highest level hypercube as a segment of R^d space, the distance between nodes can be defined by any metric in R^d space. The geometry of *HyCube* has one more property - the set of positions (coordinates) in each dimension is treated as on a ring. That means that after the point $2^l - 1$, point 0 is located. The network should therefore not be treated as a d-dimensional space limited in each dimension by 0 and $2^l - 1$ but as a d-dimensional torus with the perimeter equal to 2^l in each dimension. This modification will be important in determining distances between nodes - in every dimension the distance is determined like on a ring - the shorter of the distances in either direction is chosen. In order to obtain a short average path length and a reasonable size of the routing tables, in *HyCube*, the number of dimensions is 4 and the number of levels is 32 (to obtain a 128-bit address space).

2.2 Routing Tables

Primary Routing Table. The primary routing table has the same structure as in *Plaxton mesh*. It has l levels (the number of hierarchy levels). At each level

there are 2^d cells (d - the number of dimensions). In the routing table of a node X in the cell j at level i ($i \geq 0$), a reference is stored to a node that is in the same hypercube at level $i+1$ and in the hypercube corresponding to the number j at level i (lower level). At each level $i > 0$, there is a cell corresponding to the hypercube in which the node X exists. This cell is empty as the routing table contains a whole level corresponding to it.

Secondary Routing Table. The secondary routing table of a node X contains nodes from adjacent hypercubes to the hypercube of node X in each dimension, in both directions, at each level. An adjacent hypercube is one whose coordinate in the particular dimension is greater or smaller by 1 than the coordinate of the hypercube of X at the particular level (taking into consideration passing coordinate 0 - like on a ring). The secondary routing table does not contain nodes in cells at the highest level as hypercubes corresponding to them are included in the primary routing table. Also, one of the sibling hypercubes on each level in each dimension is covered by the primary routing table.

The secondary routing table gives a higher level of flexibility in the next hop selection. If the distance between nodes is defined by a metric in R^n space, it is very likely that the secondary routing table contains nodes that are closer to any arbitrarily chosen node.

Neighborhood Set (Closest Neighbors Set). Beside two routing tables described above, nodes maintain sets of closest to them (according to the chosen metric) nodes existing in the network. These sets allow to route messages (decreasing the distance left), even if there are no appropriate nodes in both routing tables. Such sets greatly increase the probability of delivering a message in the presence of node failures. In *HyCube*, the size of this set is 16.

The neighborhood set should provide a possibility to route packets regardless of the direction in which the destination node is located. Therefore, it is important that nodes in neighborhood sets be evenly distributed in respect of directions. There might be a scenario where some nodes would have more neighbors in one direction and no neighbors in other directions. In such a case, the nodes would not be able to route packets in all directions (using neighborhood sets). The problem becomes more significant in the presence of node failures.

Ensuring even distribution of nodes in respect of directions may cause some more distant nodes to be included in neighborhood sets and pass over some closer nodes. Thus, both, proximity and even distribution, should be considered.

The technique of ensuring even distribution of nodes adopted by *HyCube* splits the space into parts (quadrants of the system of coordinates with the center in the node whose neighborhood set is considered) and attempts to ensure that in each part, the number of nodes is the same. Nodes are chosen to the particular parts by proximity.

2.3 Routing

In each routing step, the cells from the primary routing table are selected that correspond to nodes that share at least one group of d bits longer prefix of id

with the destination node *id* than with the current node *id*. In a hierarchical hypercube, these cells correspond to hypercubes in which the destination node is located, at lower levels than the lowest level hypercube containing both, the current and the destination node.

If no node is found, the cells from the secondary routing table are selected that correspond to sibling hypercubes in the direction in which the destination node is located, in each dimension, at levels $\lfloor \log_2 d_{dim} \rfloor$ and $\lceil \log_2 d_{dim} \rceil$, where d_{dim} is the distance from the current node to the destination node in the dimension *dim* (the distance between the coordinates on the ring). Only those nodes that share at least one d-bit group longer prefix of *id* with the destination node or share the same number of d-bit groups but are closer to the destination than the current node (according to the chosen metric) should be considered.

If still no nodes are found, all nodes from both routing tables and the neighborhood set that share at least one d-bit group longer prefix with the destination node or share the same number of d-bit groups but are closer to the destination than the current node (according to the chosen metric) are considered.

From the set of the nodes selected in the steps above, the node sharing the longest prefix of *id* with the destination (number of d-bit groups) is chosen. If there is more than one such node, the node closest to the destination (according to the routing metric in use) is chosen.

In the final part of a route, when a packet is relatively close to the destination node, the routing algorithm may omit some nodes that are close to the destination, but do not share the same long or longer prefix of *id* with the destination node than with the current node. The problem becomes more significant if a multidimensional metric is used. In *HyCube*, before choosing the next hop, each node checks if the distance to the destination is shorter than the average distance to the nodes in the neighborhood set multiplied by the factor λ: $d_{dest} < avg(d_{neigh}) \cdot \lambda$. If this condition is satisfied, all further nodes on the route are chosen based only on their distance to the destination node. They might not share the same long or longer prefix of the identifier. At first, nodes from the secondary routing table at levels $\lfloor \log_2 d_{dim} \rfloor$ and $\lceil \log_2 d_{dim} \rceil$ are checked. If no nodes are found, all nodes from both routing tables and the neighborhood set are checked. The greater the value of λ, the longer parts of routes will be determined based only on the distance left. This value should be large enough to ensure a high probability of packet delivery. However, too large values of λ would cause an increase in path lengths. The value $1,35$ was determined experimentally and is a good compromise between the path length and the probability of packet delivery.

The expected route length is $\log_{2^d} N$ hops and, on average, $\log_{2^d} N \cdot (2^d - 1)$ cells are populated in the primary routing table and $(\log_{2^d} N - 1) \cdot d$ in the secondary routing table, where N is the number of nodes in the network[2].

3 Metric

The choice of the metric, i.e. how distances between nodes are determined, has a great impact on the average route length and the probability of packet delivery.

[2] Based on the assumption that nodes are distributed evenly in the space.

Choosing a one-dimensional metric allows the use of sequential neighbors, which greatly improves the static resilience. If the number of sequential neighbors is s, half are predecessors and half are successors of the node, the packet would be dropped only if all $s/2$ nodes in the appropriate direction failed. However, the use of sequential neighbors may cause a significant increase in path lengths in the case of node failures, when many routing table cells are empty. In *HyCube*, a multidimensional metric is used (a metric in a multidimensional space), which significantly decreases the expected path length when routing using only neighborhood sets. The expected path length equals $\sqrt[d]{N} \cdot \frac{\sqrt{d}}{2}$, while for sequential neighbors it is proportional to N. This fact becomes very important when considering network maintenance algorithms. By using a multidimensional metric, the network loses properties connected with existence of sequential neighbors. However, the use of the variable metric adopting the Steinhaus transform yields better static resilience than with the use of sequential neighbors.

Let us consider routing using only neighborhood sets and assume that in each step, the next hops are chosen only by the distance left to the destination, without ensuring the prefix condition. Only some part of the neighborhood set nodes is closer to the destination node than the current node. It is crucial that the number of such nodes be as large as possible (so even in the case of many node failures the packet will not be dropped). We may estimate the expected ratio of the number of nodes to which the packet may be routed to the number of all nodes in the neighborhood set as the probability that the packet may be routed to a single node in the neighborhood set.

The most common metrics in R^n space are Minkowski distances:

$$L_m(x,y) = \left(\sum_{i=0}^{d-1} |x_i - y_i|^m \right)^{\frac{1}{m}}, m \geq 1 \tag{1}$$

In particular, L_1 is the Manhattan (taxicab) distance, L_2 is the Euclidean distance and L_∞ is the Chebyshev distance. L_2 is the only metric from the Minkowski distances that preserves distances between nodes after space rotation. This causes some undesirable properties of metrics L_1 and L_3 to L_∞. When these metrics are used, if the destination node is located in different directions, the expected numbers of nodes in the neighborhood set to which the packet may be routed are different. For this reason, only L_2 was considered.

If the Euclidean metric is used, the probability that a packet may be routed to a node in the neighborhood set may be calculated as a function of $k = \frac{d}{r}$, where d is the distance left to the destination node and r is the distance from the neighborhood set node to the current node. It is the ratio of the number of points that are in the distance r from the current node and are closer to the destination than d, to the number of all points being at the distance r from the current node. To simplify, the numbers of points were determined as lengths of the curves, areas of the surfaces and their equivalents for spaces with more dimensions. The calculated values of the probability that a node in the neighborhood set is closer to the destination (for varying numbers of dimensions) are presented in Fig. 2. The figure shows that the number of nodes in the neighborhood set, to which

a packet may be routed, strongly depends on k. The closer the packet is to the destination node, the fewer appropriate nodes. The situation gets worse as the number of dimensions increases. This fact has a great impact on static resilience - fewer node failures may cause the packet to be dropped.

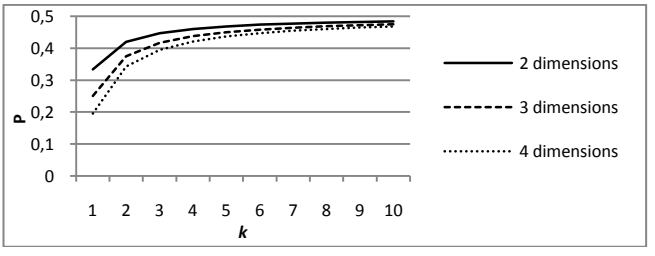


Fig. 2. Probability that a node in the neighborhood set is closer to the destination node than the current node

In [8], the Steinhaus transform was described. The theorem presented says that if X is a set and D is a metric in this set, D' is also a metric in X for any $a \in X$, where:

$$D'(x,y) = \frac{2D(x,y)}{D(x,a) + D(y,a) + D(x,y)} \tag{2}$$

Applying a metric with the Steinhaus transform for every route, setting a to the *id* of the source node, causes the next hops to be chosen in such a way that they are closer to the destination node and more distant from the source node. Such a solution increases the expected number of neighborhood set nodes to which packets can be routed in each step - it allows sending packets using more roundabout routes, but still being convergent to the destination node.

The use of the Steinhaus transform yields very good routing parameters and very high static resilience for networks containing relatively few nodes. For networks containing much more nodes, the addend $D(x,a)$ in the denominator, where x is the current node, has less influence on the value of the distance as its changes in particular steps are very minor compared with the value of the entire denominator. Thus, the more nodes in the network, the lesser the influence of the Steinhaus transform on the static resilience. However, some modification can be introduced - a variable metric adopting the Steinhaus transform, where point a is changed by nodes on routes. Point a is initially set to the source node *id*. The following nodes, before choosing the next hops, check whether they are closer (in terms of the Euclidean metric) to the destination than the current point a. In such a case, point a is changed and gets the value of the current node. Such a way of changing point a ensures that the routing is convergent to the destination (there will be no cycles on routes) and gives a high level of flexibility in the next hop selection along the whole route, regardless of the network size. The expected route length is still proportional to $\sqrt[d]{N}$ and owing to the increase in the flexibility in the next hop selection, static resilience is even better than in

networks using sequential neighbors, which can be seen in the simulation results presented in the remainder of the paper. Figure 3 presents a comparison of simulation results for different metrics for a 4-dimensional network containing 1000 nodes. The routing algorithm simulated did enforce the common *id* prefix length condition. For comparison, if packets were routed using sequential neighbors, the curve would keep at about 0.5, regardless of the distance left.

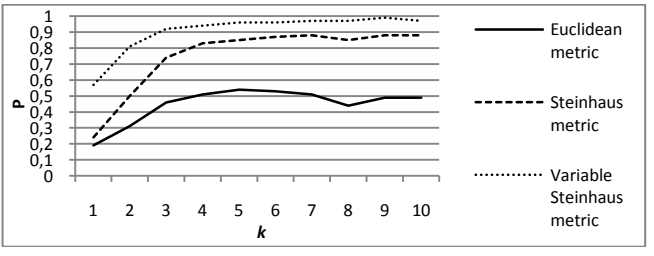


Fig. 3. Probability that a packet may be routed to a node in the neighborhood set

The final steps of routing with the use of a metric with the Steinhaus transform may cause a packet to be sent to a node that is more distant from the destination than it would be if the Steinhaus transform was not applied. When a node on a route cannot find the next hop in its routing tables and neighborhood set, it is possible that the route is broken in a point that is not the closest one to the destination in terms of the Euclidean metric. In some cases, it is crucial to reach the closest possible node if the destination node is not reached. Thus, one more modification was introduced to *HyCube* - when a packet cannot be routed by a node, the node tries to route it based only on the Euclidean distance left to the destination. All next hops after that should be chosen in the same way. Such an approach will cause that in the case the packet is dropped, a relatively close node to the destination is reached. From the experiments (for a network containing 1000 nodes, with 50% failed nodes, routing using only neighborhood sets), it appears that applying this phase in routing allows packets to be sent to a closer node in 79% cases and also increases the static resilience of the network.

4 Experimental Results

This section presents experimental results obtained in simulations of *HyCube* and *Pastry*. The properties evaluated were the static resilience and the average route length (based only on successful routes). For this purpose, a network generator and a simulator were implemented. The network generator generates a random network (*HyCube* or *Pastry*) containing the requested number of nodes - random node identifiers and random nodes in routing tables. Neighborhood sets are generated according to the criteria described in Section 2.2. The simulator reads the network generated by the network generator and simulates routing

between random pairs of nodes in the presence of varying numbers of random node failures. A node failure means removing the node from the network and from the routing tables of all nodes maintaining a reference to it.

Figure 4 presents the results of the simulations of static resilience of *HyCube* containing 1000 and 10000 nodes with the use of different metrics. It can be seen that for the network containing 10000 nodes, the influence of the Steinhaus transform is very little. However, the variable Steinhaus metric gives a significant increase in the successful path rate, regardless of the network size.

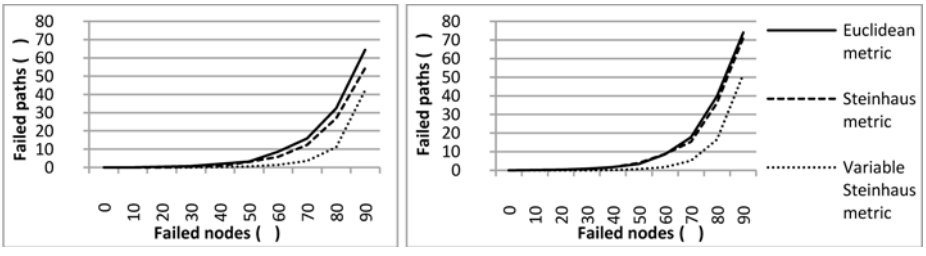


Fig. 4. Static resilience of *HyCube* - 1000 nodes (left) and 10000 nodes (right)

Figure 5 presents a comparison of simulation results of *HyCube* and *Pastry* networks containing 10000 nodes. In the simulations, the secondary routing table was not used (because it does not exist in *Pastry*). The figure shows that *HyCube* is more resilient to node failures and the average route length is shorter than in *Pastry*. The results prove that despite the absence of sequential neighbors, a more robust architecture was achieved.

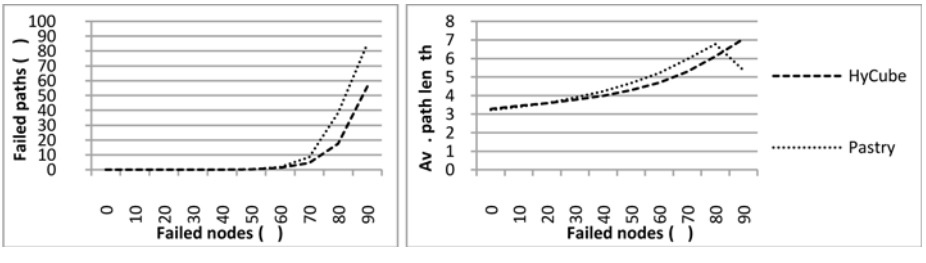


Fig. 5. Static resilience and path length increase in *HyCube* and *Pastry* (10000 nodes)

5 Conclusion

In this paper, the routing geometry and the routing algorithm of *HyCube* were presented - a DHT network based on a hierarchical hypercube geometry.

The experimental results indicate that the most crucial in terms of efficiency and static resilience was the choice of an appropriate metric and neighbor

selection algorithms (the way the nodes are chosen to the routing tables). The simulations proved that the approach presented gives shorter average path lengths in the case of many node failures in comparison to solutions using sequential neighbors. The decrease in path lengths results from the use of a multidimensional metric. However, despite the absence of sequential neighbors, a very high level of static resilience was reached, which was achieved by the use of the variable metric adopting the Steinhaus transform.

In comparison with *Pastry*, *HyCube* has better results both, in respect of the packet delivery probability and the path length increase in the presence of node failures. Moreover, *HyCube* has a very important advantage over *Pastry* - when routing using only neighborhood sets, the average path length is proportional to $\sqrt[d]{N}$, whereas when routing using only sequential neighbors, the expected path length is proportional to the number of nodes in the network.

To summarize, *HyCube* is an efficient and credible implementation of a distributed hash table. It is scalable and provides efficient routing of messages, even in the case of a large number of node failures.

References

1. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In: Proc. of the ACM SIGCOMM 2001 Technical Conference, pp. 149–160 (2001)
2. Maymounkov, P., Mazières, D.: Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, p. 53. Springer, Heidelberg (2002)
3. Rowstron, A.I., Druschel, P.: Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, p. 329. Springer, Heidelberg (2001)
4. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A Scalable Content-Addressable Network. In: Proc. of the ACM SIGCOMM 2001 Technical Conference (2001)
5. Malkhi, D., Naor, M., Ratajczak, D.: Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In: Proc. of the 21st ACM Symposium on Principles of Distributed Computing, PODC 2002 (2002)
6. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of DHT routing geometry on resilience and proximity. In: Proc. of the 2003 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications, SIGCOMM 2003 (2003)
7. Plaxton, C.G., Rajaraman, R., Richa, A.W.: Accessing Nearby Copies of Replicated Objects in a Distributed Environment. In: Proc. of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 311–320 (1997)
8. Clarkson, K.L.: Nearest-Neighbor Searching and Metric Space Dimensions. In: Nearest-Neighbor Methods for Learning and Vision: Theory and Practice. MIT Press, Cambridge (2006)

Accuracy and Performance of Single versus Double Precision Arithmetics for Maximum Likelihood Phylogeny Reconstruction

Simon A. Berger and Alexandros Stamatakis*

The Exelixis Lab, Dept. of Computer Science, Technische Universität München
Boltzmannstr. 3, 85748 Garching b. München, Germany
{bergers,stamatak}@in.tum.de
http://wwwkramer.in.tum.de/exelixis/

Abstract. The multi-core revolution and the biological data flood that is generated by novel wet-lab techniques pose new technical challenges for large-scale inference of phylogenetic trees from molecular sequence data. We present the first assessment of accuracy and performance trade-offs between single and double precision arithmetics and the first SSE3 vectorization for computing the Phylogenetic Likelihood Kernel (PLK) which forms part of many state-of-the art tools for phylogeny reconstruction and consumes 90-95% of the overall execution time of these tools. Moreover, the PLK also dominates memory consumption, which means that deploying single precision is desirable to accommodate increasing memory requirements and to devise efficient mappings to GPUs. We find that the accuracy provided by single precision is sufficient for conducting tree searches, but that the increased amount of scaling operations to prevent numerical underflow, even when using SSE3 operations that accelerate the single precision PLK by 60%, generates run-time penalties compared to double precision on medium-sized datasets. However, on large datasets, single precision can yield significant execution time savings of 40% because of increased cache efficiency and also reduces memory footprints by 50%.

Keywords: Phylogenetic inference, single versus double precision arithmetics, RAxML, Maximum Likelihood, SSE3 instructions.

1 Introduction

The emergence of many-core architectures and accelerator devices as well as the molecular data flood generated by novel high-throughput sequencing techniques require new approaches for orchestrating compute-intensive Bioinformatics kernels.

Within this context, we assess speed and accuracy trade-offs between single precision (henceforth abbreviated as SP) and double precision (henceforth

* This work is funded under the auspices of the Emmy-Noether program by the German Science Foundation (DFG).

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 270–279, 2010.

abbreviated as DP) floating point arithmetics for the Phylogenetic Likelihood Kernel (PLK [1]) that is used to reconstruct phylogenetic (evolutionary) trees from molecular sequence data.

A phylogenetic tree is an unrooted binary tree that represents the evolutionary relationships among species. The input of a phylogenetic analysis is a multiple sequence alignment comprising nucleotide or protein sequence data from organisms that are alive today. The alignment is an $n \times m$ data matrix, that contains, e.g., n DNA sequences which all have a length of m nucleotide characters (columns/sites). The output is an unrooted binary tree that represents the evolutionary history of those organisms. The tips (also called leaves or taxa) of the tree represent species alive today in contrast to internal (ancestral) nodes that represent species that have become extinct.

The PLK is one of the most widely used optimality criteria to score and thus chose among distinct evolutionary scenarios (phylogenetic trees). Many program packages are available that implement the PLK, either for standard Maximum Likelihood analyses (RAxML [2], GARLI [3]) or to conduct Bayesian phylogenetic inference (MrBayes [4], BEAST [5]). All PLK-based phylogenetic inference programs spend the largest part of overall run time (90-95%) in the computation of the likelihood function [6]. The aforementioned tools are widely used by biologists and have accumulated over 20,000 citations. Therefore, it is important to assess and devise HPC solutions for this important Bioinformatics kernel.

We present the first accuracy assessment between SP and DP arithmetics for the PLK and also exploit the usage of SSE3 instructions in the PLK. SP arithmetics can also solve memory bottlenecks in analyses of large-scale phylogenomic datasets that can already require up to 120GB of main memory under DP. The deployment of SP for the PLK can reduce memory requirements by almost 50%. SP arithmetics are also required to map the PLK onto GPUs, since at present SP arithmetics are approximately one order of magnitude faster than DP arithmetics on GPUs. We find that SP arithmetics are sufficiently accurate to conduct ML-based tree searches on trees with less than approximately 2,000 taxa and hence can be used for accelerating the kernel on Graphics Processing Units. We also achieve performance improvements of more than 40% (DP) and 60% (SP) via deployment of SSE3 instructions on general purpose CPUs. Finally, we demonstrate that SP can be used to significantly accelerate PLK computations on large phylogenomic datasets because of increased cache efficiency.

The remainder of this paper is organized as follows: In Section 2 we briefly describe how the likelihood is calculated on phylogenetic trees. Thereafter, we cover related work on floating point implementations and usage of accelerators for the PLK (Section 3). In Section 4 we describe the SSE3 and SP implementations and provide experimental results in the subsequent Section 5. We conclude in Section 6.

2 Computing the Likelihood of a Tree

The input of a standard phylogenetic analysis consists of a multiple sequence alignment with n sequences (taxa/tips) and m alignment columns. The output

is an unrooted binary tree; the n taxa are located at the leaves of the tree and the inner nodes represent common extinct ancestors. The branch lengths essentially represent the relative time of evolution between nodes in the tree. To compute the likelihood on a fixed tree topology several additional ML model parameters are required: the instantaneous nucleotide substitution matrix Q which contains the transition probabilities for time dt between nucleotide (4×4 matrix) or Amino Acids (20×20 matrix) characters. Additionally, the prior probabilities of observing the nucleotides, e.g., $\pi_A, \pi_C, \pi_G, \pi_T$ (for DNA data), and the α shape parameter that forms part of the Γ model [7] of rate heterogeneity need to be determined. The Γ model accounts for the fact that different sites evolve at different speeds. Finally, one also requires the $2n - 3$ branch lengths in the unrooted tree topology.

To compute the likelihood of a fixed *unrooted* binary tree topology given these parameters, initially one needs to compute the entries for all internal probability vectors (located at the inner nodes) that contain the probabilities $P(A), P(C), P(G), P(T)$, of observing an `A,C,G,` or `T` at each site/column c of the input alignment at the specific inner node. Those probability vectors are computed bottom-up from the tips towards a virtual root that can be placed into any branch of the tree using a procedure known as the Felsenstein pruning algorithm [1]. Under certain standard model restrictions (time-reversibility of the model) the likelihood score will be the same, regardless of the placement of the virtual root.

Every probability vector entry $\boldsymbol{L}(c)$ at a position c ($c = 1...m$) of the tips and the inner nodes of the tree topology, contains the four probabilities $P(A)$, $P(C)$, $P(G)$, $P(T)$ of observing a nucleotide `A, C, G, T`, at a specific column c of the input alignment. The probabilities at the tips (leaves) of the tree for which observed data *is* available are set to 1.0 for the observed nucleotide character at the respective position c, e.g., for the nucleotide `A`: $\boldsymbol{L}(c) = [1.0, 0.0, 0.0, 0.0]$. Given a parent node k, and two child nodes i and j (with respect to the virtual root), their probability vectors $\boldsymbol{L}^{(i)}$ and $\boldsymbol{L}^{(j)}$, the respective branch lengths leading to the children b_i and b_j, and the transition probability matrices $P(b_i), P(b_j)$, the probability of observing an `A` at position c of the ancestral (parent) vector $\boldsymbol{L}_A^{(k)}(c)$ is computed as follows:

$$\boldsymbol{L}_A^{(k)}(c) = \Big(\sum_{S=A}^{T} P_{AS}(b_i) \boldsymbol{L}_S^{(i)}(c) \Big) \Big(\sum_{S=A}^{T} P_{AS}(b_j) \boldsymbol{L}_S^{(j)}(c) \Big) \tag{1}$$

The transition probability matrix $P(b)$ for a given branch length is obtained from Q via $P(b) = e^{Qb}$. Once the two probability vectors $\boldsymbol{L}^{(i)}$ and $\boldsymbol{L}^{(j)}$ to the left and right of the virtual root (vr) have been computed, the likelihood score $l(c)$ for an alignment column c ($c = 1...m$) can be calculated as follows, given the branch length b_{vr} between nodes i and j:

$$l(c) = \sum_{R=A}^{T} \Big(\pi_R \boldsymbol{L}_R^{(i)}(c) \sum_{S=A}^{T} P_{RS}(b_{vr}) \boldsymbol{L}_S^{(j)}(c) \Big) \tag{2}$$

The overall score is then computed by summing over the per-column log likelihood scores: $LnL = \sum_{c=1}^{m} log(l(c))$.

An important property of the likelihood function is the assumption, that sites evolve independently, i.e., all entries c of the probability vectors $\boldsymbol{L}$ can be computed independently. This property represents the main source of fine-grain parallelism in the PLK [6].

In order to compute the *Maximum* Likelihood value for a fixed tree topology all individual branch lengths, as well as the parameters of the Q matrix and the α shape parameter, must also be optimized via an ML estimate. For the Q matrix and the α shape parameter the most common approach consists in using Brent's algorithm. In order to evaluate changes in Q or α the entire tree needs to be re-traversed, i.e., a *full* tree traversal needs to be conducted in order to correctly re-compute the likelihood. For the optimization of branch lengths, the Newton-Raphson method is commonly used. In order to optimize the branches of a tree, the branches are repeatedly visited and optimized one-by-one until the achieved branch length change is smaller than some pre-defined ϵ. The bulk of all of the likelihood computations consists of `for`-loops over the length m of the vectors $\boldsymbol{L}$. These `for`-loops require for instance 95% of total execution time in RAxML.

Avoiding Numerical Underflow: The methods deployed for avoiding numerical underflow represent an important implementation and performance issue. As can be derived from Formula 1, the values in the probability vectors $\boldsymbol{L}$ at the inner nodes of the tree will progressively become smaller as we approach the virtual root, since we are conducting successive multiplications with the probability values in the transition probability table P. Especially for trees with many taxa, measures need to be taken to avoid numerical underflow in the probability vectors.

The scaling in RAxML is conducted as follows: At a column c of an ancestral probability vector $\boldsymbol{L}$ we scale the entries if $\boldsymbol{L}_A(c) < \epsilon \wedge \boldsymbol{L}_C(c) < \epsilon \wedge \boldsymbol{L}_G(c) < \epsilon \wedge \boldsymbol{L}_T(c) < \epsilon$, where $\epsilon = 1/2^{256}$ for DP and $\epsilon = 1/2^{32}$ for SP. If probability vector column c at vector $\boldsymbol{L}$ needs to be scaled, we simply multiply all entries $\boldsymbol{L}_A(c), \boldsymbol{L}_C(c), \boldsymbol{L}_G(c), \boldsymbol{L}_T(c)$ by 2^{256} (DP) and 2^{32} (SP) respectively. In order to annihilate the scaling events at the virtual root we keep track of the total number of scaling events conducted per column by using integer vectors $\boldsymbol{U}$ that maintain the scaling events and correspond to the respective probability vectors. At the virtual root, given $\boldsymbol{L}^{(i)}, \boldsymbol{L}^{(j)}$ and the corresponding scaling vectors $\boldsymbol{U}^{(i)}, \boldsymbol{U}^{(j)}$ we compute the likelihood as follows:

$$l(c) = \frac{1}{2^{256}}^{U^{(i)}(c)+U^{(j)}(c)} \Big(\sum_{R=A}^{T} \big(\pi_R \boldsymbol{L}_R^{(i)}(c) \sum_{S=A}^{T} P_{RS}(b_{vr}) \boldsymbol{L}_S^{(j)}(c) \big) \Big) \tag{3}$$

If we take the logarithm of $l(c)$ and $\epsilon = 1/2^{256}$ this can be re-written as:

$$log(l(c)) = (U^{(i)}(c) + U^{(j)}(c)) log(\epsilon) + log\Big(\sum_{R=A}^{T} \big(\pi_R \boldsymbol{L}_R^{(i)}(c) \sum_{S=A}^{T} P_{RS}(b_{vr}) \boldsymbol{L}_S^{(j)}(c) \big) \Big) \tag{4}$$

Memory Requirements: The memory requirements for ML-based phylogeny programs are dominated by the space required for the inner probability vectors $\boldsymbol{L}$ and the inner scaling vectors $\boldsymbol{U}$. Depending on the memory organization and data structures used, we need to assign at least one probability vector and one scaling vector to each of the $n-2$ inner nodes of the tree. Since for the values at the leaves we only have 15 alternative probability vector entries using ambiguous DNA character encoding, we only need to store one vector $\boldsymbol{L}$ of length 15 which can then be accessed using the input sequences as index. The input sequences can be stored as simple `char` arrays. Hence, the memory requirements for computing the likelihood on a DNA alignment (without accommodating for rate heterogeneity) with n taxa and m columns requires $n \cdot m \cdot 1$ bytes for the input sequences, $(n-2) \cdot m \cdot 4 \cdot 8$ bytes for the probability vectors and $(n-2) \cdot m \cdot 4$ bytes for the scaling vectors. If we use the standard Γ model of rate heterogeneity the space requirements for the probability vectors increase to $(n-2) \cdot m \cdot 16 \cdot 8$ bytes. Hence, the memory requirements are dominated by the space required for the inner probability vectors and can be reduced by factor 2 using SP arithmetics. This is an important issue since we are receiving an increasing number of reports from RAxML users that encounter memory shortages.

3 Related Work

We are not aware of any related work that assesses accuracy and speed trade-offs between SP and DP floating point arithmetics for the PLK. However, such analyses have been conducted for standard numerical linear algebra kernels, e.g., systems of linear equations [8] where the authors propose a mixed precision approach, i.e., an initial optimization under SP and a final refinement under DP. Such a procedure that dynamically switches from SP to DP, could also be applied to phylogenetic inference, i.e., one could initially infer a rough tree structure (the big picture) under SP and then refine it under DP. However, we find that this is not necessary and that the loss of accuracy is insignificant with respect to the tree topology (see Section 5).

Nonetheless, there is some on-going work to port GARLI [3] to SP (Derrick Zwickl, personal communication) for the same reasons as RAxML. Surprisingly, no efforts have been undertaken and published with respect to deploying SSE instructions to improve performance of the PLK on new-generation x86 architectures. The only documented usage of SIMD instructions for the PLK on the CELL processor is described in [9].

MrBayes [4], which is a program for Bayesian phylogenetic inference, has been ported down to SP five years ago, mainly to better accommodate the significantly larger memory requirements caused by the multiple heated and cold Markov Chains in the Metropolis-Coupled search procedure. Bayesian floating point implementations are more straight-forward since no iterative procedures (Newton-Raphson, Brent's algorithm) are required to optimize ML model parameters. We are also not aware of any study that deals with the accuracy trade-offs regarding tree topologies in MrBayes following the transition from DP

to SP. In addition, the MrBayes source code also contains SSE3 instructions, but potential performance gains have not been documented and SSE3 does not form part of the standard distribution. Finally, increased scaling events for SP also occur in MrBayes. According to profiling runs of MrBayes using `gprof` within the framework of an OpenMP parallelization, we found that the scaling procedure requires approximately 20% of overall execution time.

The Bayesian program BEAST [5] has also recently been ported to SP in order to be mapped to a GPU (*Bioinformatics*, in press, preprint at `http://tree.bio.ed.ac.uk/publications/390/`). The porting to SP was mainly conducted for efficiently computing 60-state Codon models on GPUs for which impressive speedups of two orders of magnitude are achieved. However, the speedup obtained in comparison to a DP C implementation for DNA data between CPU and GPU is only around factor 4 since the mathematical operations that are required for a 4-state transition matrix can not be mapped as efficiently to a GPU. Moreover, the performance comparison between GPU and CPU could be improved in favor of the CPU. The code on the multi-core CPU, an Intel Core 2 Extreme with a total of 4 cores, is run sequentially and not using an OpenMP or Pthreads-based fine-grained parallelization of the PLK, i.e., a speedup of factor 4 could be achieved via multi-threading. If SSE3 instructions were deployed for the PLK, an additional two-fold speedup over the GPU could be achieved. Finally, the performance analysis is only conducted using a single 63 taxon dataset. Hence, a potential performance degradation caused by increased scaling events as more taxa are added to the alignment is not assessed.

4 Implementation

Single Precision Version: The SP version of RAxML was implemented using a similar strategy as in MrBayes. We still conduct a large portion of the numerically sensitive operations, like base frequency computations or Eigenvalue/Eigenvector decomposition that are required to compute $P(t) = e^{Qt}$ in DP and then cast the $P(t)$ matrix to SP. We also compute the derivatives of $P(t)$ that are required for conducting the Newton-Raphson procedure for branch length optimization in DP and then cast them to SP. Thus, only the main bulk of operations as outlined in Equations 1 and 2 is actually conducted under SP. Based on prior experience with several unsuccessful attempts to port RAxML to SP, this approach seems to yield the numerically most stable implementation.

Finally, we also empirically adapted (increased) various ϵ settings that determine the number of iterations in the Newton-Raphson as well as in the iterative procedures for optimization of the remaining ML model parameters Q and α. The convergence parameters were adapted in such a way that the SP version carries out approximately as many iterations for branch length and ML model parameter optimization as the DP version. These increased settings yield slightly worse likelihood scores than the DP version, but we find that this has no significant impact on the relative likelihood-based order of trees (see Section 5).

SSE3 for Likelihood Computations: We vectorize computations that are special cases of a general dense matrix multiplication; the computations on $\boldsymbol{L}^{(i)}$ and $\boldsymbol{L}^{(j)}$ in Formula 1 over all sites c and all nucleotides A, C, G, T are matrix products of the form $P \cdot \boldsymbol{L}^{(i)}$ and $P \cdot \boldsymbol{L}^{(j)}$. We assessed the usage of highly optimized ATLAS-BLAS routines [10], but because of the unfavorable matrix dimensions (multiplication of the 4×4 matrix P with the $4 \times m$ matrix $\boldsymbol{L}$) we even observed a slowdown. We also deploy the horizontal addition instructions in SSE3 for the reduction operations that are required to efficiently complete the scalar product as indicated in Formula 1.

SSE3 for Likelihood Scaling: We also vectorized the scaling procedure as outlined in Section 2 using SSE3. This is particularly important for the SP implementation, since the number of scaling events increases by one order of magnitude (see Figure 2). SSE3 instructions are used to efficiently determine the maximum value of $\boldsymbol{L}_A(c), \boldsymbol{L}_C(c), \boldsymbol{L}_G(c), \boldsymbol{L}_T(c)$ (see Section 2) and then compare the maximum to the ϵ value, thereby eliminating several conditional statements. SSE3 vectorization was implemented by inserting SSE3 intrinsics into the C code, rather than via inline assembly. This leaves room for further optimizations of the instruction schedule and register allocation by the compiler. It is important to note that, the Intel `icc` compiler (v 11.1) is not able to vectorize the `for`-loops of the PLK, despite numerous attempts to re-write the loops for facilitating auto-vectorization.

5 Experimental Setup and Results

To assess accuracy of the SP versus the DP version we initially generated collections of "good" trees that are encountered during a tree search under DP on 9 single-gene real-world DNA datasets containing 150 up to 1,908 taxa. Thereafter, we applied the RAxML function for scoring a set of trees (`-f n` option, for details please refer to the RAxML Manual) under the standard GTR+Γ model of nucleotide substitution to score the respective tree collections under SP and DP. The absolute likelihood values are not important for a tree search, but only the relative scores, i.e., how well can our SP implementation discriminate between alternative trees via the likelihood value. To this end, we computed the Spearman rank correlation coefficient ρ between the likelihood-based tree rankings obtained via DP and SP tree evaluations to assess if SP provides a sufficient degree of accuracy. We also used the above experiments to obtain execution times for the SP and DP versions, as well as for the SSE3-based and standard versions of the code. As test platform we used a SUN x4600 multi-core system equipped with 32 AMD Opteron cores running at 2.7GHz and a total of 64GB of RAM. We used `gcc` (v. 4.3.2) to compile all versions of the code.

In Table 1 we provide the number of taxa in the test datasets (column: # taxa), the number of trees in the respective tree collections (column: # trees), the Spearman correlation between SP and DP likelihood-based tree orders (column: ρ), the speedup between the standard SP and DP versions (column: S/D), between the SSE3-based SP and DP versions (column: S-SSE3/D-SSE3),

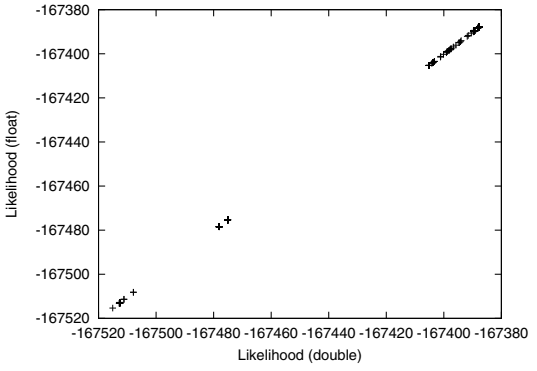


Fig. 1. Scatter plot of SP and DP tree scores for a datasets with 1,604 sequences

between the standard and SSE3-based SP versions (column: S-SSE3/S) and between the standard and SSE3-based DP versions (column: D-SSE3/D). Overall, the order of trees induced by SP and DP likelihood scores is highly correlated and hence SP suffices to conduct full tree searches. In Figure 1 we provide a scatter plot of SP versus DP likelihood scores for the worst-case Spearman coefficient of 0.95 on the alignment with 1,604 taxa. The average slowdown of SP over DP is approximately factor 2, but improves to 1.5 for the respective SSE3 implementations. This improvement is due to the higher speed gains of approximately 60% in the SP SSE3 implementation compared to about 40% in the DP SSE3 implementation.

Table 1. Test datasets, number of trees in tree collections, Spearman coefficients, and speedup values between all code versions

# taxa	# trees	ρ	SP/DP	SP-SSE3/DP-SSE3	SP-SSE3/SP	DP-SSE3/DP
150	200	0.99	1.39	0.83	0.36	0.61
218	260	0.99	2.58	1.71	0.42	0.64
354	160	0.97	1.44	0.78	0.36	0.67
500	300	0.99	2.31	1.62	0.44	0.63
628	180	0.95	1.39	0.80	0.36	0.62
714	320	0.99	2.41	1.70	0.43	0.61
1,512	520	0.99	2.63	1.82	0.44	0.63
1,604	220	0.95	2.38	1.69	0.40	0.63
1,908	360	0.99	1.29	0.78	0.40	0.66

In Figure 2 we outline the number of scaling events over the number of taxa (note the log scale on the y-axis) for the evaluation of a single tree. As mentioned before, the SP version requires about one order of magnitude *more* scaling operations. In Figure 3 we provide corresponding execution times (note the log scale on the y-axis) for the evaluation of a single tree using the standard SP and

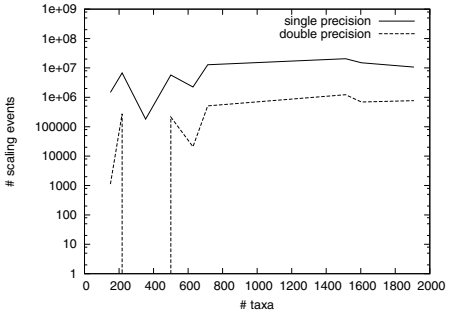


Fig. 2. Number of scaling events for SP and DP versions over the number of taxa

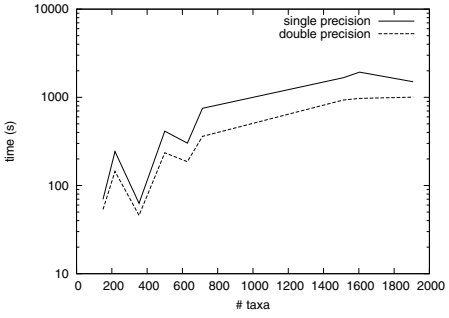


Fig. 3. Execution times in seconds for SP and DP versions over the number of taxa

DP versions of the code. The similar shapes of the two curves clearly show that scaling operations dominate SP run times.

In a second set of experiments we measured execution times and inference accuracy for the Pthreads-based RAxML code on a large-scale phylogenomic protein dataset with 321,145 distinct alignment patterns and 232 taxa which requires approximately 40GB of RAM under DP and the Γ model of rate heterogeneity. On a 16-core SUN x4600 system (code without SSE3) we found that an inference under SP is 40% faster and yields an equally good final tree, when scored under DP, while reducing the memory requirements to 20GB. On a 32-core SUN x4600 system (code with SSE3) we found that the SP-based code using the CAT approximation of rate heterogeneity [11] is three times faster than a standard Γ-based inference under DP, yields a slightly better final likelihood score, and only requires 5GB of main memory.

Finally, we also assessed numerical stability on alignments containing $\geq$ 2,000 taxa and found that the code encounters problems with numerical stability on such large alignments under SP. Similar observations were made by Derrick Zwickl (personal communication) on his SP implementation of GARLI, i.e., problems with numerical stability on many-taxon datasets seems to be a general problem.

6 Conclusion

We have presented the first thorough assessment of accuracy and speed trade-offs with respect to using SP versus DP floating-point arithmetics for the Phylogenetic Likelihood Kernel in a Maximum Likelihood framework. In addition, we have conducted the first SSE3-based vectorization of the PLK. Our results indicate that SP can be deployed to accurately infer phylogenetic trees with less than 2,000 taxa. In addition, SP arithmetics significantly reduce memory requirements of large phylogenomic analyses and substantially improve inference times via increased cache efficiency. Thereby, in combination with the CAT approximation of rate heterogeneity, we can design tools that enable large-scale phylogenomic

inference "for the masses" by significantly reducing computational resource requirements. We also find that, SSE3 yields significant run time improvements; we achieve 60% for SP and approximately 40% for DP. Thus, users can chose between the DP-SSE3 and SP-SSE3 versions for DNA or protein data in the current RAxML release (v. 7.2.3), depending one their memory and CPU time constraints as well as on the alignment shape.

Future work, will cover a more detailed analysis of the numerical stability on many-taxon datasets as well as work on a GPU version of RAxML.

References

1. Felsenstein, J.: Evolutionary trees from DNA sequences: a maximum likelihood approach. J. Mol. Evol. 17, 368–376 (1981)
2. Stamatakis, A.: RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. Bioinformatics 22(21), 2688–2690 (2006)
3. Zwickl, D.: Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion. PhD thesis, University of Texas at Austin (April 2006)
4. Ronquist, F., Huelsenbeck, J.: MrBayes 3: Bayesian phylogenetic inference under mixed models. Bioinformatics 19(12), 1572–1574 (2003)
5. Drummond, A., Rambaut, A.: BEAST: Bayesian evolutionary analysis by sampling trees. BMC Evol. Biol. 7(214), 1471–2148 (2007)
6. Ott, M., Zola, J., Aluru, S., Stamatakis, A.: Large-scale Maximum Likelihood-based Phylogenetic Analysis on the IBM BlueGene/L. In: Proc. of IEEE/ACM Supercomputing Conference 2007, SC 2007 (2007)
7. Yang, Z.: Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites. J. Mol. Evol. 39, 306–314 (1994)
8. Kurzak, J., Dongarra, J.: Implementation of mixed precision in solving systems of linear equations on the Cell processor. Concurrency and Computation 19(10), 1371 (2007)
9. Blagojevic, F., Nikolopoulos, D.S., Stamatakis, A., Antonopoulos, C.D.: RAxML-Cell: Parallel Phylogenetic Tree Inference on the Cell Broadband Engine. In: Proc. of International Parallel and Distributed Processing Symposium, IPDPS 2007 (2007)
10. Whaley, R., Dongarra, J.: Automatically tuned linear algebra software (ATLAS). In: Proc. Supercomputing, vol. 98 (1998)
11. Stamatakis, A.: Phylogenetic Models of Rate Heterogeneity: A High Performance Computing Perspective. In: Proc. of IPDPS 2006. HICOMB Workshop, Proceedings on CD, Rhodos, Greece (April 2006)

Automated Design of Assemblable, Modular, Synthetic Chromosomes

Sarah M. Richardson[1,2], Brian S. Olson[3], Jessica S. Dymond[1,4], Randal Burns[6], Srinivasan Chandrasegaran[5], Jef D. Boeke[1,4], Amarda Shehu[3], and Joel S. Bader[1,7]

[1] High Throughput Biology Center, Johns Hopkins University School of Medicine, Baltimore, MD 21205, USA
[2] McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins University School of Medicine, Baltimore, MD 21205, USA
[3] Department of Computer Science, George Mason University, Fairfax, VA 22030
[4] Department of Molecular Biology and Genetics, Johns Hopkins University School of Medicine, Baltimore, MD 21205, USA
[5] Department of Environmental Health Sciences, Johns Hopkins University Bloomberg School of Public Health, Baltimore, MD 21205, USA
[6] Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218
[7] Department of Biomedical Engineering, Johns Hopkins University, Baltimore MD 21218

Abstract. The goal of the *Saccharomyces cerevisiae* v2.0 project is the complete synthesis of a re-designed genome for baker's yeast. The resulting organism will permit systematic studies of eukaryotic chromosome structure that have been impossible to explore with traditional gene-at-a-time experiments. The efficiency of chemical synthesis of DNA does not yet permit direct synthesis of an entire chromosome, although it is now feasible to synthesize multi-kilobase pieces of DNA that can be combined into larger molecules. Designing a chromosome-sized sequence that can be assembled from smaller pieces has to date been accomplished by biological experts in a laborious and error-prone fashion. Here we pose DNA design as an optimization problem and obtain optimal solutions with a parallelizable dynamic programming algorithm.

1 Introduction

Synthetic biology requires careful design of nucleotide sequences. Practical applications of synthetic biology include modifying proteins through amino acid changes and redesigning multi-gene pathways. Synthetic biology enables the study of genes that are difficult to manipulate by traditional means, such as ancestral genes inferred from phylogenetic studies of extant biological sequences.

Design tools for synthetic DNA sequences have been limited primarily to gene-length sequences. For example, GeneDesign assists gene editing and synthesis at the physical level of nucleotides and oligos that can be ordered and used in-house for inexpensive gene assembly [1]. Other software packages perform logical-level checks of the syntax and grammar of well-formed transcriptional units,

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 280–289, 2010.

such as whether genes have promoters, start codons, non-inhibitory secondary structures, and properly located termination sequences [2,3].

While full genomes are an attractive synthesis target, scaling up synthetic routes from genes (thousands of nucleotides) to genomes (millions of nucleotides) is proving difficult [4]. Some groups have avoided this problem by focusing on genomes so small that they can be put together exactly as though they were merely very large genes, as the Endy group did when refactoring part of the 39 kilobase (kb) genome of bacteriophage T7 [5]. This approach is limited to viral genomes [6] or bacterial plasmids and is impractical for groups interested in larger prokaryotes or eukaryotes. Other groups are using a top-down approach, which involves taking an existing genome and editing it in place as the Blattner group did for *Escherichia coli* [7], with edits usually limited to deletions rather than insertions or substitutions. Alternatively, a bottom-up approach was used to synthesize the entire bacterial *Mycoplasma genitalium* genome [8]. Overlapping oligos were assembled into larger cassettes, combined in yeast into a complete genome. Ultimately it must be transplanted as a complete genome into the host cell, which has been achieved for natural but not synthetic DNA [9]. The obligate parasite *M. genitalium* has a single 582 kb circular genome, much smaller than the 4.7 megabase (Mb) *E. coli*. The bottom-up approach is unlikely to scale to larger synthetic targets and delays integration and testing to the very end.

Our group has launched an effort to synthesize the genome of the yeast *Saccharomyces cerevisiae*. Yeast has a 12 Mb genome comprising sixteen linear chromosomes, ten of which are each larger than the entire genome of *M. genitalium*, and the smallest of which is five times larger than the genome of T7. As a eukaryote, yeast has more chromosomal features than viral and bacterial genomes. Our synthetic target is an edited version of the native yeast genome that includes the biological equivalent of debug statements that will allow us to identify and remove the DNA-equivalent of dead code and probe the cryptic function of non-protein-coding regulatory sequences, many of which await discovery by methods such as ours.

The synthetic strategy combines bottom-up synthesis with in-place editing, which permits it to scale to whole chromosomes and genomes. In a hierarchical procedure, DNA sequences of 60 nucleotides are ordered and combined using PCR and genetic engineering into larger synthetic pieces that can then be introduced into yeast to replace cognate wild-type sequences through homologous recombination. The experimental workflow is sufficiently streamlined for adaptation to undergraduate teaching laboratories [10]. This strategy should therefore be a valuable addition to the field of synthetic biology.

Our synthetic strategy, and genetic engineering in general, requires the use of restriction enzymes to cut and recombine DNA molecules at enzyme-specific recognition sites, also termed restriction or cut sites. Requirements for the occurrence of suitable restriction sites introduce constraints on any target sequence we wish to synthesize. If a synthetic target does not satisfy the constraints, it is possible to modify the target by editing its DNA sequence, at the cost of

possibly introducing unwanted and unpredictable changes to biological function. Restriction enzymes have varying prices, and less expensive enzymes often perform better. Furthermore, solutions with roughly equal spacing between cut sites are preferred to solutions with widely varying spacings.

Optimal design has been a practical problem for our in-house project because the combinatorial complexity prevents human experts from reliably generating acceptable error-free solutions, and there are too many sub-optimal solutions for a brute-force computational enumeration. Our design requires that a chromosome be split into 10 kb chunks delimited by restriction enzyme cut sites. We permit the chunk boundaries to vary within a 1000 nt window to include protein-coding regions where we can introduce restriction enzyme cut sites by synonymous recoding of protein-coding sequences. The yeast genome is 70% protein-coding, and each 1 kb interval contains approximately 700 protein-coding nucleotides, or codons for 233 amino acids. Due to genetic code redundancy, a restriction enzyme with a 6-bp recognition site will usually recognize at least 6 different amino acid pairs (three reading frames, two directions). The probability that 6 pairs out of the 20^2 total possible pairs are absent in a 233 amino acid sequence is approximately $e^{-233\times 6/400}$, or 3%. This theoretical expectation that each possible 6-cutter is individually feasible for each cut site is borne out in practice. With over 100 restriction enzymes with 6-nt recognition readily available, there are over 100^{26} possible solutions for a small yeast chromosome. A larger chromosome requiring 130 cut sites could have a search space of 100^{130} solutions. We found that top-down, greedy approaches are inefficient because feasible solutions are sparse, and even algorithms with long look-aheads lead to dead ends.

Here we formulate optimal design as a constrained optimization problem for a combined cost reflecting the edit distance and the efficiency of the synthetic strategy. We present an algorithm that computes the optimal solution. The algorithm uses a parallelizable indexing step to catalog all edits that are unlikely to affect gene function (i.e., those that involve purely synonymous base substitutions). Next, the algorithm uses dynamic programming to divide the search into optimal sub-problems that can be computed in parallel. It uses dead-end elimination to remove sub-optimal paths from the search tree. A recent human design for 90 kb assisted with available computational tools required over 40 man-hours of work. Scaling to the yeast genome would require nearly 3 years of this dedicated expert. In contrast, our implementation takes 2.5 minutes for the same 90 kb region, roughly a 1000× speed-up, and only 5 to 6 hours for the entire genome. Furthermore, the algorithm produces output that is superior to all of our expert-generated results, allowing us to quickly create several plans of action for inspection and evaluation — and perhaps concurrent synthesis. The algorithm is implemented in Perl and Python for compatibility with our existing synthetic biology software and for access to core I/O and visualization functionality from the Generic Model Organism Database project [11].

2 Biological Constraints

Restriction enzymes are proteins that cleave DNA at exquisitely predictable recognition sites. We use a set of 121 commercially-available enzymes from REBASE [12], a restriction enzyme database. Genetic engineering techniques often require that an enzyme has one, and only one, recognition site in a large piece of DNA. Sites must therefore be rare (to avoid multiple cuts) but not too rare (to avoid the absence of a site altogether). Synthetic biology offers the ability to manipulate the distribution of recognition sites in target DNA molecules by recoding the native sequence to add or remove recognition sites. That is, we can place $n-1$ restriction enzyme recognition sites very precisely in our synthetic sequence, and then, rather than synthesizing a single long molecule, we can synthesize and assemble n shorter molecules, cut them with the appropriate enzymes, and ligate them together into the long molecule we wish to obtain.

For instance, the enzyme BamHI will cut any double stranded DNA molecule that contains its recognition site 5′ GGATCC 3′, striking asunder both strands between G and GATCC: $\overset{5'\ \mathtt{G}|\mathtt{GATC}\ 3'}{3'\ \mathtt{C}\overline{\mathtt{CTAG}|}\mathtt{G}\ 5'}$. The two double-stranded cleavage products can be ligated back to their original partners in a process called re-ligation, or ligated to other pieces of DNA that have been cut with enzymes that leave the same "overhang". In the case of BamHI, cleavage leaves two DNA molecules, both with overhangs that read 5′ GATC 3′. Because this overhang sequence is palindromic (note that palindromes in DNA are distinct from natural language due to antiparallel DNA strands), the two species of cleavage products may ligate to themselves or to each other, yielding a mix of three ligation products. Mixed products are undesirable because they reduce the yield of the desired product and produce potentially inhibitory or deleterious side-products. Enzymes leaving non-palindromic overhangs are much more desirable because they ensure a single canonical product from an assembly reaction. Our algorithm filters enzymes by relevant criteria, including price, but most importantly, by the kind of overhang left after cleavage. Of the 121 available enzymes, 55 are capable of generating non-palindromic cleavage sites. Not all enzymes are alike in efficiency, availability, or price, and the algorithm uses a real-valued cost to rank the overall performance of each enzyme.

The algorithm also requires a chromosome whose protein-coding regions are annotated. The annotations are used to enforce a hard constraint imposed by our limited knowledge of biology; we have implemented a design rule that all DNA changes to introduce or remove restriction sites must be accomplished within protein-coding regions. Edits to non-protein-coding DNA are not permitted because intergenic and intronic sequences house yet-uncharacterized regulatory sequences that could be disrupted by a single base change. Algorithms to evaluate the cost of different edits therefore require knowledge of the boundaries between introns (sequences transcribed from DNA to RNA but then spliced out), exons (sequences remaining in the processed RNA transcript including all protein-coding sequences), and intergenic sequences (DNA sequence that is not transcribed but which may contain important regulatory or structural features).

Feasible edits to protein-coding sequences are achieved by substituting synonymous codons, three-nucleotide sequences that encode the same amino acid. Synonymous edits exploit the redundancy of the genetic code. Isolated edits that leave the amino acid sequence unchanged usually do not affect protein expression or activity. Widespread edits, however, are more likely to change RNA secondary structure or affect protein activity by changing the speed of translation, which can in turn alter protein levels or generate mis-folded protein. When genes overlap, edits must retain the amino acid sequences of both proteins.

We use the standard Generic Feature Format (.gff), which consists of tab-delimited entries annotating features followed by actual nucleotide sequence in FASTA format. GFF files for many different organisms are easily obtained from public genome repositories on the Internet; the yeast chromosomes were taken from the Saccharomyces Genome Database [13].

3 Data Collection and Indexing

Once the algorithm is provided with a list of enzyme recognition sites and an annotated chromosome sequence, it generates a database of every single restriction enzyme site in the chromosome. Every intergenic sequence is parsed for existing recognition sites. Those that are found are treated as immutable; they may be used as boundaries if they are fortuitously placed, but nothing can be done to improve their placement or the overhangs they leave. A suffix tree is created from all possible 6-frame translations of restriction enzyme recognition sites, such that each node in the tree is an amino acid string that may be reverse translated to be a recognition site. Every exonic sequence in the chromosome is then searched with the suffix tree both for existing recognition sites and for sites where a recognition site could be introduced without changing the protein sequence of the gene using. As long as they occur within protein coding genes, existing and potential recognition sites may be manipulated to yield any of several different overhangs, all of which are computed by the algorithm. It is necessary that every extant site be indexed so that later in the algorithm, when potential sites are considered, the number of sites that must be modified is a known contribution to the cost. The construction of the enzyme database may be executed in parallel because parsing each gene and intergenic sequence for restriction sites is completely independent; no processing element working on a region need communicate with any other processing element. We implemented this collection step in multi-threaded Perl for compatibility with existing code.

4 Landmark Selection

The current implementation of our global assembly scheme requires restriction enzyme sites as landmarks that divide the chromosome into segments of about 10 kb, which may then be built up from oligos [10]. To enforce hierarchical modularity, some of these landmarks will have additional uniqueness requirements. Landmark 1 sites will be 10 kb landmarks that can also divide the chromosome

into 100 kb segments; landmark 2 sites will be 10 kb landmarks that can also divide the chromosome into 30 kb segments. Every other 10 kb landmark is a landmark 3 site (Fig. 1). The innermost landmark 3 sites are not permitted to occur anywhere else within their flanking landmark 1 or 2 sites; likewise, the landmark 2 sites may not appear anywhere within their flanking landmark 1 sites, and any consecutive landmark 1 sites may not be the same. The overhangs left by any two consecutive landmark sites should be different and non-palindromic. These constraints prevent enzymes from cutting the DNA at unwanted locations or yielding a mix of ligation products from cross-reactions.

The goal is to select the optimal permutation of landmark enzymes for a full complement of evenly spaced landmark sites. From now on we will refer to a permutation of restriction enzyme recognition sites and locations as a plan. A valid plan must meet the adjacency and uniqueness constraints described above and is then given a real-valued cost reflecting editing costs and penalties. The optimization goal is to identify the plan with the lowest cost.

Each landmark separates two regions of size L ($L = 100$ kb for landmark 1 sites, 30 kb for landmark 2 sites, and 10 kb for landmark 3 sites). The algorithm checks the database for all eligible sites E within the upstream and downstream regions, a window of $2L$. The cost for enzyme E at location X is

$$\text{Cost}(E) = \ln \text{Price}(E) + A \times \max[0, \frac{|X - X_0|}{L} - \Delta] + B_0 n_0 + B_1 n_1 + B_2 n_2. \quad (1)$$

The optimal position of the landmark, X_0, is at the midpoint of the $2L$ window. In order to penalize changes to critical genes more heavily, counts of edited genes are categorized as non-essential (n_0), slow growth (n_1), and essential (n_2). The cost of the enzyme in US dollars per unit is Price(E). We use the parameters $A = 15$, $\Delta = 0.1$, $B_0 = 0.1$, $B_1 = 0.6$, and $B_2 = 1.1$. This objective function was selected to match generally with the intuition of biological experts. To keep costs on a uniform scale, enzyme prices are sensitive to fold-ratios and optimal positions are calculated relative to an allowed 10% variation. The fixed parameter values, and even the entire form of the objective function, are open to re-analysis subsequent to experimental tests of designed sequences.

4.1 Dynamic Programming

Brute-force enumeration of all valid plans fails because the time to enumerate grows exponentially as $O(m^n)$, where m is the number of possible enzyme choices per landmark (55 commercially available enzymes capable of leaving non-palindromic overhangs) and n is the number of landmarks required (scaling linearly with chromosome size with one landmark per 10,000 nucleotides). For yeast chromosomes, n ranges from approximately 25 to 180. We have developed an efficient approach that employs dynamic programming and dead-end elimination to run in $O(nm^6)$ time.

Since the chromosome is divided into 100 kb regions by landmark 1 sites (Fig. 1), the optimal plan for the chromosome additively combines the optimal plans for each of these regions. Finding the optimal plans for each region are

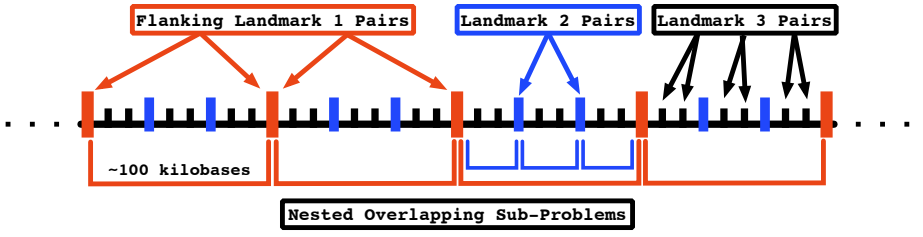


Fig. 1. Each sub-problem is bounded by a pair of flanking landmark 1 sites

overlapping subproblems due to the landmark 1 sites shared by consecutive regions. Optimal sub-structure and overlapping sub-problems are the hallmarks of dynamic programming [15].

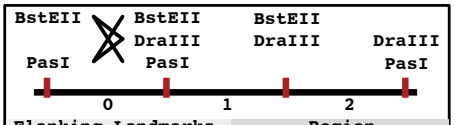
A few iterations of the algorithm's progress through the dynamic programming matrix are displayed (Fig. 2). Columns 0 to $n-1$ correspond to regions. Rows correspond to pairs of flanking landmark 1 enzymes. Each cell in the matrix maintains the optimal cost, opt-cost(i, j), for a particular row i and a particular column j. Each cell is initialized with an '`x`' to represent an invalid or infinite cost. The costs are computed as

$$\text{opt\_cost}(i, j) = \text{cost}(i, j) + \min_{\text{compatible}i'}\{\text{opt\_cost}(i', j-1)\} \tag{2}$$

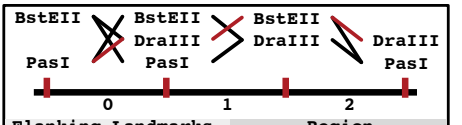
where cost(i, j) refers to the cost associated with a particular region j and a particular pair of enzymes i, and compatible i' refers only to those rows i' that are compatible with row i. Compatibility here refers to the flanking constraint: the enzyme pair (e_1, e_2) at column $j-1$ is compatible with the enzyme pair (e_3, e_4) at position j if and only if $e_2 = e_3$. The cost associated with the optimal plan for the chromosome will be found in the cell with the minimum value in the last column of the matrix. As in classic dynamic programming, the plan associated with the minimum cost can be recovered by tracing back from the optimum.

Each sub-problem requires computing the optimal cost (i, j) for a given region and a given selection of enzymes for the flanking landmark 1 sites. A brute force approach to the sub-problems yields a runtime of $O(nm^{10})$, where the exponent 10 refers to the total number of landmark sites: two for landmark 1, two for landmark 2, and six for landmark 3 (Fig. 1). The complexity of the problem becomes $O(nm^2)$ plus the cost of pre-computing the sub-problems. As we explain below, the complexity is actually bounded by the computation of the sub-problems. For this reason pre-computing the sub-problems is ideal as it facilitates parallelization.

Sites for landmarks 2 and 3 can be grouped into pairs, and a dynamic programming algorithm similar to the one described for landmark 1 sites can then be applied. The order of enzymes in a sub-plan matters for the cost, but not for the uniqueness constraints. Therefore the algorithm need only store the cost associated with combinations rather than permutations of the sub-plans. This



Flanking Landmarks	Region 0	1	2
(BstEII,DraIII)	-1	x	x
(BstEII,PasI)	-2	x	x
(DraIII,BstEII)	x	x	x
(DraIII,PasI)	x	x	x
(PasI,BstEII)	-4	x	x
(PasI,DraIII)	-2	x	x



Flanking Landmarks	Region 0	1	2
(BstEII,DraIII)	-1	-7	-11
(BstEII,PasI)	-2	x	-6
(DraIII,BstEII)	x	-5	x
(DraIII,PasI)	x	-4	-10
(PasI,BstEII)	-4	x	x
(PasI,DraIII)	-2	-3	x

Fig. 2. A small example of the dynamic programming algorithm. All cells were initialized as null. The optimal path is displayed in red. Cost is as computed by Eq. 2

yields an overall running time of $O(m^6)$ for each sub-problem, since each sub-problem contains m^2 choices for the flanking landmark 1 pair, m^2 choices for the landmark 2 pair, and $3m^2$ choices for the landmark 3 pairs. Furthermore, the number of sub-problems is linear with respect to the size of the chromosome, making the over-all running time of our dynamic programming algorithm

$$O(\text{sub} - \text{problems} + \text{global} - \text{problem}) = O(nm^6 + nm^2) = O(nm^6) \qquad (3)$$

4.2 Dead-End Elimination

Although $O(nm^6)$ time improves on exponential scaling, it is still inconvenient for $m = 55$ possible restriction enzymes. We have developed a dead-end elimination algorithm to reduce the effective size of m.

We observed that landmarks often have enzyme choices that are not constrained by any other sites (Fig. 3). In this example, landmark site 9 may use enzymes AccI and BanI regardless of which enzymes are chosen for sites 8, 10, 11, 14, and 17. Since AccI and BanI are independent, either choice may be used regardless of all other landmarks. Therefore, there is no need to keep any enzyme choice that has a cost greater than the minimum of cost(AccI) and cost(BanI) (in this case AccI). For any landmark plan that uses BanI, DrdI, or FokI in landmark site 9, we can always improve the cost by substituting AccI; using BanI, DrdI, or FokI is guaranteed to produce a sub-optimal result.

In practice, the dead-end elimination algorithm reduces the size of of m by 25% or more on the yeast chromosomes considered in this work, speeding the run time by a factor of $(4/3)^6$ or about 5.6.

4.3 Parallel Implementation

Since the performance of the enzyme selection algorithm is bounded by the pre-processing of independent sub-problems, we used the PP (Parallel Python)

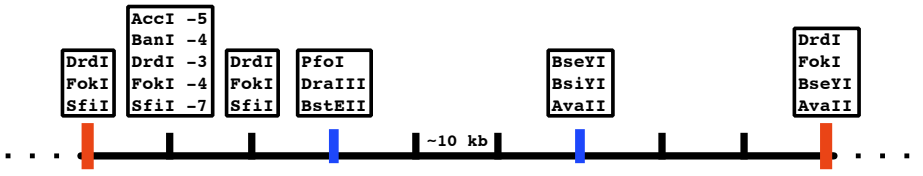


Fig. 3. The enzyme choice for site 9 is constrained by the choices for 8, 10, 11, 14, and 17. Using Dead-end Elimination, we can prune BanI, DrdI, and FokI from site 9.

module to facilitate parallelization across multiple independent machines. This module allows seamless addition of processor cores across multiple physical systems. This approach scales up to n cores, where n is the number of landmark 1 delimited regions in the chromosome. Each landmark 1 region is sent to a different core. For the biggest yeast chromosome, the run time decreased from 155 sec to 30 sec, a 5× speedup, using 16 cores in a cluster of quad-core 2.6 GHz Opteron processors with 4GB of RAM each. In practice, the speed-up from parallelization is limited by the region with the highest effective m value — the single region that takes the most time to compute.

5 Results and Discussion

Manual design of even the smallest yeast chromosome is a laboriously slow and error-prone process that does not guarantee an optimal solution. We have transformed this synthetic biology design problem into a formal optimization problem which we solve with an efficient implementation using suffix-tree indexing, dynamic programming, and dead-end elimination. Our approach produces optimal designs with a 1000× speed advantage over human experts. The input is an annotated target chromosome and list of restriction enzyme specifications and costs. The output is a minimally edited synthetic target, and existing downstream software can convert this into oligonucleotides for ordering from a vendor.

Designs generated by this algorithm are currently being synthesized and experimentally tested as part of a project to create a yeast cell with synthetic DNA [10]. We hope that our code will be useful to others planning similar projects at any scale. All source code is available from the authors' website, www.baderzone.org, under an open source BSD license.

Acknowledgments. We thank Pamela Meluh, Yan Qi, and John Kloss for careful reading of the manuscript and discussion. S.M.R. was supported by Department of Energy Computational Science Graduate Fellowship DE-FG02-97ER25308. This project was supported in part by a Microsoft Research Award to J.S.B and J.D.B., and grant MCB 0718846 from the National Science Foundation to J.D.B, J.S.B and S.C.

References

1. Richardson, S.M., Wheelan, S.J., Yarrington, R.M., Boeke, J.D.: GeneDesign: rapid, automated design of multikilobase synthetic genes. Genome Res. 16, 550–556 (2006)
2. Cai, Y., Hartnett, B., Gustafsson, C., Peccoud, J.: A syntactic model to design and verify synthetic genetic constructs derived from standard biological parts. Bioinformatics 23, 2760–2767 (2007)
3. Villalobos, A., Ness, J.E., Gustafsson, C., Minshull, J., Govindarajan, S.: Gene Designer: a synthetic biology tool for constructing artificial DNA segments. BMC Bioinformatics 7, 285–293 (2006)
4. Czar, M.J., Anderson, J.C., Bader, J.S., Peccoud, J.: Gene synthesis demystified. Trends Biotechnol. 27, 63–72 (2009)
5. Chan, L.Y., Kosuri, S., Endy, D.: Refactoring bacteriophage T7. Mol. Sys. Bio. 1 (2005), doi: 10.1038/ msb4100025
6. Cello, J., Paul, A.V., Wimmer, E.: Chemical synthesis of poliovirus cDNA: generation of infectious virus in the absence of natural template. Science 297, 1016–1018 (2002)
7. Pósfai, G., Plunkett, G., Fehér, T., Frisch, D., Keil, G.M., Umenhoffer, K., Kolisnychenko, V., Stahl, B., Sharma, S.S., Arruda, M., Burland, V., Harcum, S.W., Blattner, F.R.: Emergent properties of reduced-genome. *Escherichia coli*. Science 312, 1044–1046 (2006)
8. Gibson, D.G., Benders, G.A., Andrews-Pfannkoch, C., Denisova, E.A., Baden-Tillson, H., Zaveri, J., Stockwell, T.B., Brownley, A., Thomas, D.W., Algire, M.A., Merryman, C., Young, L., Noskov, V.N., Glass, J.I., Venter, J.C., Hutchison, C.A., Smith, H.O.: Complete chemical synthesis, assembly, and cloning of a *Mycoplasma genitalium* genome. Science 319, 1215–1220 (2008)
9. Lartigue, C., Glass, J.I., Alperovich, N., Pieper, R., Parmar, P.P., Hutchison, C.A., Smith, H.O., Venter, J.C.: Genome transplantation in bacteria: changing one species to another. Science 317, 632–638 (2007)
10. Dymond, J., Scheifele, L., Richardson, S.M., Lee, P., Chandrasegaran, S., Bader, J.S., Boeke, J.D.: Teaching Synthetic Biology, Bioinformatics, and Engineering to Undergraduates: The Interdisciplinary Build-a-Genome Course. Genetics 181, 13–21 (2009)
11. Stein, L.D., Mungall, C., Shu, S., Caudy, M., Mangone, M., Day, A., Nickerson, E., Stajich, J.E., Harris, T.W., Arva, A., Lewis, S.: The Generic Genome Browser: A Building Block for a Model Organism Database. Genome Res. 12, 1599–1610 (2002)
12. Roberts, R.J., Vincze, T., Posfai, J., Macelis, D.: REBASE–enzymes and genes for DNA restriction and modification. Nucl. Acids Res. 35, D269–D270 (2007)
13. Fisk, D.G., Ball, C.A., Dolinski, K., Engel, S.R., Hong, E.L., Issel-Tarver, L., Schwartz, K., Sethuraman, A., Botstein, D., Michael, C.J.: *Saccharomyces cerevisiae* S288C genome annotation: a working hypothesis. Yeast 23, 857–865 (2006)
14. Mattson, T.G., Sanders, B.A., Massingill, B.L.: Patterns for Parallel Programming, 1st edn. Addison-Wesley, Reading (2004)
15. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. McGraw-Hill, New York (2001)

GPU Parallelization of Algebraic Dynamic Programming

Peter Steffen[1,2], Robert Giegerich[1], and Mathieu Giraud[2]

[1] Technische Fakultät, Universität Bielefeld, D-33501 Bielefeld, Germany
psteffen@techfak.uni-bielefeld.de, robert@techfak.uni-bielefeld.de
[2] CNRS, LIFL, Université Lille 1, 59 655 Villeneuve d'Ascq cedex, France
mathieu.giraud@lifl.fr

Abstract. Algebraic Dynamic Programming (ADP) is a framework to encode a broad range of optimization problems, including common bioinformatics problems like RNA folding or pairwise sequence alignment. The ADP compiler translates such ADP programs into C. As all the ADP problems have similar data dependencies in the dynamic programming tables, a generic parallelization is possible. We updated the compiler to include a parallel backend, launching a large number of independent threads. Depending on the application, we report speedups ranging from 6.1× to 25.8× on a Nvidia GTX 280 through the CUDA libraries.

1 Introduction

Dynamic programming in bioinformatics. In biological sequence analysis, there arise numerous combinatorial optimization problems that are solved by dynamic programming. Pattern matching in DNA or protein sequences, comparison for local or global similarity, and structure prediction from RNA sequences are frequent tasks, as well as the modeling of families of proteins and RNA structures with the widely used Hidden Markov Models (HMMs) and stochastic context free grammars (SCFG), respectively [5]. The scoring schemes associated with these optimization problems can be quite sophisticated. The thermodynamic model for RNA structure prediction, for example, has more than thousand parameters. This requires elaborate case analysis. Objective functions often ask for more than a single answer, such as the best non-overlapping pattern hits to a genome above a certain score threshold. Finally, biological sequences tend to be long (from 77 characters for a tRNA, 10000 for a gene, $3 * 10^6$ for a bacterial genome, to the $3 * 10^9$ nucleotides of a mammalian genome such as human or mouse). The time and space requirements for a dynamic programming algorithm are often limiting factors for the problems the biologists need to solve. The development of reliable and efficient dynamic programming algorithms in bioinformatics is a recurring challenge, in sharp contrast to the simplicity suggested by the textbook examples of dynamic programming which we use to teach computer science students.

Algebraic dynamic programming. In all these optimization problems, the logical problem decomposition follows the decomposition of the input sequence

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 290–299, 2010.

into subwords. It has been observed early that the resulting dynamic programming recurrences strongly resemble those of a Cocke-Younger-Kasami [3] type parsing algorithm [20]. Pursuing this analogy, we have developed an algebraic style of dynamic programming (ADP) over sequential data. The search space of the optimization problem at hand is described by a *yield grammar*, which is a regular tree grammar generating a tree language, and implicitly a context-free language as the set of leaf sequences of these trees. Scoring and optimization are described by an *evaluation algebra*, which interprets the tree operators as functions that compute local score contributions, and hence solve larger problems when given optimal solutions of smaller ones, consistent with the general paradigm of dynamic programming. This leads to a complete specification of dynamic programming algorithms on a rather high level of abstraction.

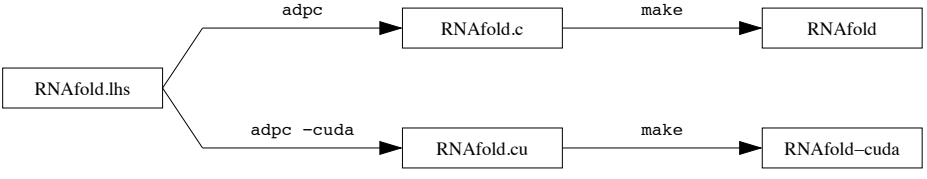


Fig. 1. ADP workflow. The goal of this study is to conceive and implement an automatic GPU parallelization (bottom).

General-purpose computation on GPU. For a few years, issues with heat dissipation have prevented the processors from having higher frequencies. One of the answers to maintain the Moore's Law is the use of parallel processing with massively manycore architectures. Graphic processing units (GPUs) are a first step towards those architectures, and recent trends blur the line between such GPUs and multi-core processors.

GPUs were used in bioinformatics since 2005 for phylogenetic studies [4], then for multiple sequence alignment based on an optimized Smith-Waterman implementation [10]. The CUDA libraries, first released in 2007 [2], have deeply simplified the development on GPUs. Recent papers provide speedups on applications involving suffix trees [19] or again Smith-Waterman comparisons [9,11,13], error correction in DNA short-reads sequencing [21], computation with position weight matrices [8], RNA folding [18], and neighbor-joining trees for multiple sequence alignments [12].

The current Nvidia architectures [2] offer two levels of parallelism. For the coarse-grained level, several multiprocessors execute *blocks* of independent computations. Each multiprocessor is then a kind of large SIMD device, able to process several different fine-grained *threads* at a given time. All those threads are executing exactly the same instructions: if a *divergence* in a condition occurs, the branches of the condition are serialized.

Contents. In this paper we describe an approach to extend the ADP compiler such that it generates parallel programs (Figure 1). This approach has been

implemented in the ADP compiler with the CUDA libraries [2]. Our new contribution is thus a generic method to create parallel CUDA programs for bioinformatics applications – classical and yet-to-be written ones. The next section presents background on Algebraic Dynamic Programming. Section 3 presents the GPU parallelization of ADP. Section 4 gives some results and discussion: depending on the application, we get speedups ranging from 6.1× to 25.8× on a Nvidia GTX 280.

2 Algebraic Dynamic Programming

We briefly introduce the Algebraic dynamic programming (ADP) methodology on a simple Nussinov type RNA secondary structure prediction problem, the maximization of the number of base pairs [15]. Section 4 reports results for several other applications. See [6] for a complete presentation of the ADP method.

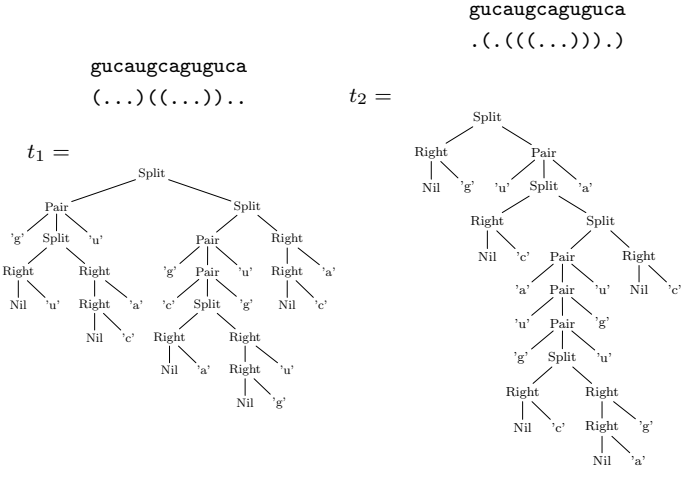


Fig. 2. Two candidates in the search space for the best secondary structure for the sequence `gucaugcaguguca`

When designing a dynamic programming algorithm in algebraic style, we need to specify four constituents: the input alphabet, the search space, the scoring of the candidates, and the objective function.

Alphabet. The input RNA sequence is a string over the alphabet $\mathcal{A} = \{a, c, g, u\}$.

Search space. Given an input sequence $w \in \mathcal{A}^*$, the search space is the set of all possible secondary structures the sequence w can form. In the ADP terminology, the elements of the search space for a given input sequence are called *candidates*.

Figure 2 gives example of candidates for RNA folding. This tree representation of candidates exists for any application of dynamic programming [23]. To describe the candidates, the ADP methodology uses the notion of *tree grammar*. Figure 3 shows the grammar `nussinov78`, origin of our two example trees. For each sequence $w \in \mathcal{A}^*$, the grammar defines a search space $P_{\mathcal{G}}(w)$ that is the set of all parses of the sequence w for $\mathcal{G}$.

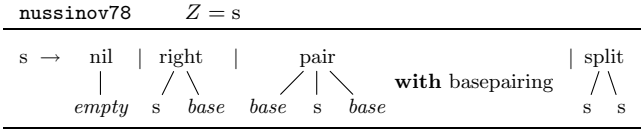


Fig. 3. Tree grammar `nussinov78` consists of one production with four alternatives. Symbol Z denotes the axiom of the grammar.

Scoring and objective. Given an element of the search space as a tree $t \in P_{\mathcal{G}}(w)$, we need to score this element. In our example we are only interested in counting base pairs, so scoring is very simple: the score of a tree is the number of *pair*-nodes in t. For the two candidates of Figure 2 we obtain scores of 3 (t_1) and 4 (t_2). Moreover, we need to choose one or several solutions from the pool of candidates. For this purpose we add an objective function h which chooses one or more elements from a list of candidate scores.

```
bpmax = (nil, right, pair, split, h) where
nil(s)        = 0
right(s,b)    = s
pair(a,s,b)   = s + 1
split(s,s')   = s + s'
h([])         = []
h([s_1,...,s_r]) = [max_{1≤i≤r} s_i]
```

Scoring schemes with objective functions are called *evaluation algebras* in ADP. The above example is the evaluation algebra `bpmax` for maximizing the number of base pairs. The flexibility of the algebraic approach lies in the fact that we don't have to stop with definition of *one* algebra: simply define another algebra and get other results for the same search space. We use the notation $\mathcal{E}(t)$ to indicate the value obtained from t under evaluation with algebra $\mathcal{E}$. All that is left to do is to evaluate the candidates in a given algebra, and make our choice via the objective function h. For example, candidates t_1 and t_2 of Figure 2 are evaluated by algebra `bpmax`, with $h(\texttt{bpmax}(t_1), \texttt{bpmax}(t_2)) = [\max(3,4)] = [4]$.

This example was fairly simple: complete RNA folding algorithms are typically based on energy minimization, and include energies of stacking regions (or helices), bulge loops, internal loops, hairpin loops and multiple loops. Figure 4 shows an excerpt of the real `RNAfold.lhs` grammar that includes the full Turner model [14]. The grammar can be read as a standard context-free grammar. The

operator `~~~` connects succeeding symbols and the operator `|||` divides alternative productions for a non-terminal. The symbol `<<<` denotes application of an algebra function and `...` denotes application of the evaluation function `h`. Finally, the operator `with` denotes the use of a filter function, that means that `(base ~~~ closed ~~~ base) 'with' basepairing` is only successful if the two bases can form a base pair.

```
rnafold alg f = axiom struct where
 (sadd,cadd,is,sr,hl,bl,br, il, il11, il12, il21, il22,
  dl, dr, dlr, edl, edr, edlr, drem, cons, ul, pul, addss, ssadd, nil, combine, h) = alg

  struct        = tabulated (sadd <<< base     ~~~ struct |||
                             cadd <<< initstem ~~~ struct |||
                             nil  <<< empty ... h)

  initstem = tabulated (is <<< loc ~~~ closed ~~~ loc ... h)
  closed   = tabulated (stack ||| ((hairpin ||| leftB ||| rightB ||| iloop ||| multiloop)
                 'with' stackpairing) ... h)

  stack     = (sr <<< base ~~~ closed ~~~ base)  'with' basepairing ... h
  hairpin   = hl <<< base ~~~ base ~~~ (region 'with' (minsize 3))   ~~~ base ~~~ base ... h
  leftB     = bl <<< base ~~~ base ~~~ region   ~~~ initstem   ~~~ base ~~~ base ... h
  rightB    = br <<< base ~~~ base ~~~ initstem ~~~ region     ~~~ base ~~~ base ... h
  iloop     = il <<< base ~~~ base ~~~ (region  'with' (maxsize 30)) ~~~ closed ~~~
                                       (region  'with' (maxsize 30)) ~~~ base ~~~ base ... h
```

Fig. 4. Excerpt from the ADP grammar `RNAfold.lhs`. The complete grammar includes further productions for multiloop structures.

3 Automatic Parallelization of ADP

Principle. A compiler that translates ADP programs into C was previously developed [7]. This task includes some advanced optimization techniques, see [22] for a detailed overview. With the option `-cuda`, the compiler is now switched into the CUDA code generation mode. The compiler uses the same backend both for CPU and GPU code generation and only differs in the following parts:

1. The dynamic programming tables need to be stored both on the host and on the global memory of the GPU. The compiler generates code to synchronize these tables.
2. For each dynamic programming table, the compiler generates a calculation function. This is the same function both for CPU and GPU mode, so the only change is that it is declared to be executed as a GPU kernel. Figure 5 shows the CUDA code for the dynamic programming main loop in the RNA secondary structure prediction program. The kernel function, `calc_all`, contains the calls for the calculation of the six dynamic programming tables.
3. In CPU mode, all table elements are calculated sequentially with increasing subword length. This order of computation has to be changed to enable parallelization. For the RNA secondary structure prediction program, the calculation of a table element (i, j) depends only on the elements in the triangle in the lower left (see Figure 6, on the left). So all elements in one

diagonal can be calculated in parallel. The whole dynamic programming table is then calculated in a loop over all diagonals (see Figure 5). This approach can be generalized to all dynamic programming algorithms over sequence data: in all ADP grammars, all results are combined from results of shorter subsequences. Therefore, the calculation of a table element (i, j) depends only on results that lie between the indices (i, j).

All these changes are done automatically by the compiler and do not require any changes to the ADP grammar. The number of blocks and threads used for the calculation can be configured as a parameter at runtime.

```
__global__ static void calc_all(int diag, int n) {
   int i = blockIdx.x*blockDim.x+threadIdx.x;
   int j = i + diag;
   if ((i <= n) && (j <= n)) {
      calc_closed(i, j);
      calc_initstem(i, j);
      calc_struct(i, j);
      calc_block(i, j);
      calc_comps(i, j);
   }
}

static void mainloop() {
   for (int diag=0; diag<=n; diag++) {
      (...)
      calc_all <<< grid, threads >>> (diag, n);
   }
}
```

Fig. 5. Kernel and main CUDA code for the dynamic programming main loop corresponding to the grammar shown on Figure 4. Note that each kernel thread also computes inner loops for the folding calculations. Actual codes are available on the ADP website (`http://bibiserv.techfak.uni-bielefeld.de/adp/cuda.html`).

Window mode. It does not make any sense to fold a complete genome as a single RNA molecule. This remark is the same for other applications: for example, a thermodynamic matcher [17] looks for some small sub-sequences (50 to some hundreds bases) matching a given structural pattern in a large sequence. In those applications, we need to compute only some diagonals above the main diagonal (Figure 6, on the right). The option `-cudaw` sets the ADP compiler in window mode. Whereas the CPU version sequentially computes all windows, the GPU version loads a large sequence into the GPU and launches a large number of independent threads, thus increasing the parallelism. This is also done automatically by the compiler and does not require any changes to the source program.

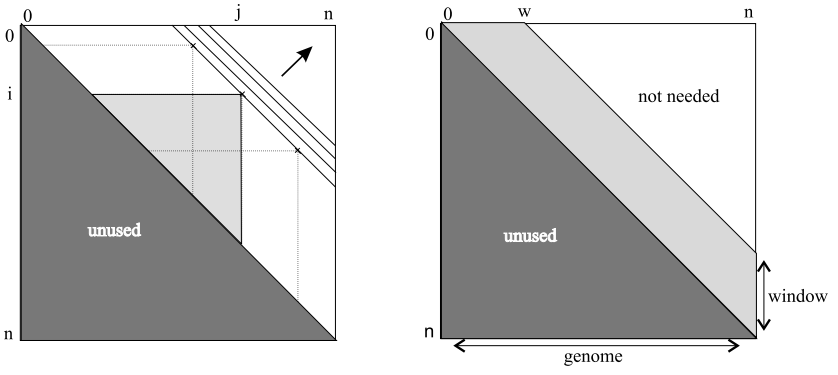


Fig. 6. Left: data dependencies for RNA secondary structure prediction. The computation of the table element (i, j) needs the $O((j-i)^2)$ elements in the underlying triangle. Right: window mode. With a large genome (of size n), we just need to fold sequences on small windows (of size w).

4 Results

Table 1 shows the results on three different applications with RNA sequences: *RNAfold* (see previous section), *pknotsRG* (detection of pseudo-knots [16]), and a tRNA thermodynamic matcher. The program *pknotsRG* predicts RNA secondary structures including a restricted class of pseudoknots. The thermodynamic matcher was created by the graphical tool *Locomotif* [17]. Practical 9.9×, 14.5× and 6.1× speedups are obtained on those real applications with a GTX 280. In these speedups, the main bottlenecks are in memory transfers, as only the global memory of the GPU is used.

In the original `RNAfold.lhs` grammar, a part of production `calc_closed` is in fact computed for only 6 out of the 16 possible basepairs (filter `‘with‘ stackpairing` on Figure 4). This brings a large divergence between the threads and breaks the GPU SIMD model. To confirm this fact, we tested a special version of this grammar, `RNAfold-bp.lhs`, that computes for every basepair the full recurrence equations (penalizing non-pairs): the speedup with the GTX 280 is almost doubled. This indicates that a similar speedup would result for the calculation of stochastic grammars, since here arbitrary base pairs are considered.

It should be noted that our speedups are lower than the best possible ones. For example, Rizk and Lavenier [18] developed an optimized GPU RNAfold implementation: in particular, they pack together the 6/16 computations corresponding to the production `calc_closed`. They obtain a 17× speedup on a GTX 280 against one core of a 2.66 GHz Xeon (applied on a whole sequence, without window mode), whereas our best speedup without window mode is only 2.8× (results not shown). However, as our approach is generic, it can be applied on several algorithms with few efforts to the user.

Table 1. Time (real times, in seconds) for executing different ADP grammars. CPU versions are compiled with `adpc`, and executed on a 2.4 GHz Core2 processor (PC1, 1 core used). and on a 3.0 GHz Xeon X5472 processor (PC2, 1 core used). GPU CUDA versions are compiled `adpc -cudaw`, and executed on a GeForce 8800 (PC1) and on a GTX 280 (PC2). Because of its increased number of cores and of its better handling of uncoalesced memory loads, the GTX 280 gives better speedups than the GeForce 8880. Note that the performance of the CPU does not impact the times reported for the GPU versions. For example, for `RNAfold`, the 19.22s for the PC1 GPU include only 0.20s of non-kernel computations, mainly for traceback in the DP matrix. Tests on `RNAfold` and `tRNA-matcher` were done on the 160 kbp genome of Candidatus Carsonella ruddii (Genbank reference `NC_008512`). Tests on `pknotsRG` were done on the first 20 kbp of the same genome.

Grammar, window size, time complexity			PC1 Core2 + GeForce 8800			PC2 Xeon + GTX 280		
			CPU	GPU	speedup	CPU	GPU	speedup
RNAfold-bp.lhs	`-w 80`	$O(w^2n)$	176.09	19.22	9.1×	133.77	5.18	25.8×
RNAfold.lhs	`-w 80`	$O(w^2n)$	43.43	8.08	5.4×	35.57	3.59	9.9×
tRNA-matcher.lhs	`-w 100`	$O(w^2n)$	52.46	6.76	7.8×	43.60	3.01	14.5×
pknotRG.lhs	`-w 80`	$O(w^3n)$	26.82	10.64	2.5×	23.54	3.25	7.2×
pknotRG.lhs	`-w 160`	$O(w^3n)$	188.68	87.65	2.2×	166.27	27.22	6.1×

On *pknotsRG*, runs with $w = 160$ get a little smaller speedup than with $w = 80$. As w is fixed, this does not limit the scalability of our approach: the input data size, n, can always grow with the same speedup.

Current limitations. Whereas grammars involving several sequences can be encoded in the ADP formalism, the ADP compiler now only works for one input sequence. Removing this limitation would allow to study other dynamic programming problems, as for example Smith-Waterman sequence alignment or RNA co-folding [6]. Finally, for some grammars (including the tRNA matcher), the ADP automatic table design generates some recursive functions, and those functions cannot be compiled with the CUDA libraries (there is no stack on the current cards). This automatic table design is removed through omitting the `-cto` option, but, in this case, the grammar should specify precisely which symbols of the grammar are to be "tabulated".

5 Perspectives

We implemented a parallel GPU CUDA backend for the ADP compiler, which works out-of-the-box for several grammars dealing with RNA sequences. The new ADP compiler and some example codes are available on the ADP website (`http://bibiserv.techfak.uni-bielefeld.de/adp/cuda.html`). Our approach is generic and requires few efforts to the user, even if the speedups are not the best ones that could be obtained by manually optimized implementations. We plan to remove the limits explained above. Other perspectives include the following points.

Shared memory. The ADP compiler could be improved to better use the memory hierarchy of the card. In the Nvidia architecture, a 16 KB *shared* memory is available for the threads in the same block. This local memory is very fast and should be used to maximize the efficiency, for example in storing portions of some dynamic programming tables. This memory is not used in our current implementation. Of course, the best usage of the shared memory depends on the application: for now, we did not find a generic way to determine from the grammar this best usage. Some hints given in the grammar file could indicate to the compiler which dynamic programming tables should be handled in this way.

Static evaluation of grammars. We plan to test other grammars, in bioinformatics as in other domains. Which grammars are efficient to parallelize, and why?

Other targets. We plan to test the ADP methodology on other manycore architectures, in particular through the new OpenCL standard [1]. Again, the fact that the ADP methodology is generic allows to write portable solutions.

Acknowledgements

Part of this research was done during P. Steffen's stay in Université Lille 1. This research was carried through the "NVIDIA Professor Partnership" program.

References

1. The Khronos Group, OpenCL 1.0 specification (2008)
2. Nvidia CUDA programming guide 2.0 (2008)
3. Aho, A.V., Ullman, J.D.: The Theory of Parsing, Translation and Compiling. Prentice-Hall, Englewood Cliffs (1973), I and II
4. Charalambous, M., Trancoso, P., Stamatakis, A.: Initial experiences porting a bioinformatics application to a graphics processor. Adv. in Informatics, 415–425 (2005)
5. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological Sequence Analysis. Cambridge University Press, Cambridge (1998)
6. Giegerich, R., Meyer, C., Steffen, P.: A discipline of dynamic programming over sequence data. Science of Computer Programming 51(3), 215–263 (2004)
7. Giegerich, R., Steffen, P.: Challenges in the compilation of a domain specific language for dynamic programming. In: Proceedings of the 2006 ACM Symposium on Applied Computing (2006)
8. Giraud, M., Varré, J.-S.: Parallel position weight matrices algorithms. In: International Symposium on Parallel and Distributed Computing, ISPDC 2009 (2009)
9. Ligowski, L., Rudnicki, W.: An efficient implementation Smith-Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases. In: IEEE International Workshop on High Performance Computational Biology, HiCOMB 2009 (2009)

10. Liu, W., Schmidt, B., Voss, G., Müller-Wittig, W.: GPU-ClustalW: using graphics hardware to accelerate multiple sequence alignment. In: Robert, Y., Parashar, M., Badrinath, R., Prasanna, V.K. (eds.) HiPC 2006. LNCS, vol. 4297, pp. 363–374. Springer, Heidelberg (2006)
11. Liu, Y., Maskell, D., Schmidt, B.: CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. BMC Research Notes 2(1), 73 (2009)
12. Liu, Y., Schmidt, B., Maskell, D.: Parallel reconstruction of neighbor-joining trees for large multiple sequence alignments using CUDA. In: IEEE International Workshop on High Performance Computational Biology, HiCOMB 2009 (2009)
13. Manavski, S.A., Valle, G.: CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. BMC Bioinformatics 9(Suppl. 2), S10 (2008)
14. Mathews, D., Sabina, J., Zuker, M., Turner, D.: Expanded sequence dependence of thermodynamic parameters improves prediction of RNA secondary structure. Journal of Molecular Biology 288, 911–940 (1999)
15. Nussinov, R., Pieczenik, G., Griggs, J.R., Kleitman, D.J.: Algorithms for loop matchings. SIAM J. Appl. Math. 35, 68–82 (1978)
16. Reeder, J., Giegerich, R.: Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. BMC Bioinformatics 5(104) (2004)
17. Reeder, J., Reeder, J., Giegerich, R.: Locomotif: From graphical motif description to RNA motif search. Bioinformatics 23(13), 391–400 (2007)
18. Rizk, G., Lavenier, D.: GPU accelerated RNA folding algorithm. In: Using Emerging Parallel Architectures for Computational Science / International Conference on Computational Science, ICCS 2009 (2009)
19. Schatz, M.C., Trapnell, C., Delcher, A.L., Varshney, A.: High-throughput sequence alignment using graphics processing units. BMC Bioinformatics 8, 474 (2007)
20. Searls, D.B.: Linguistic approaches to biological sequences. CABIOS 13(4), 333–344 (1997)
21. Shi, H., Schmidt, B., Liu, W., Mueller-Wittig, W.: Accelerating error correction in high-throughput short-read DNA sequencing data with CUDA. In: IEEE International Workshop on High Performance Computational Biology, HiCOMB 2009 (2009)
22. Steffen, P.: Compiling a Domain Specific Language for Dynamic Programming. PhD thesis, Bielefeld University (2006)
23. Steffen, P., Giegerich, R.: Versatile and declarative dynamic programming using pair algebras. BMC Bioinformatics 6(224) (2005)

Parallel Extreme Ray and Pathway Computation

Marco Terzer and Jörg Stelling

Biosystems Science and Engineering, ETH Zürich, CH-8092 Zürich, Switzerland
{marco.terzer,joerg.stelling}@bsse.ethz.ch

Abstract. Pathway analysis is a powerful tool to study metabolic reaction networks under steady state conditions. An Elementary pathway constitutes a minimal set of reactions that can operate at steady state such that each reaction also proceeds in the appropriate direction. In mathematical terms, elementary pathways are the extreme rays of a polyhedral cone—the solution set of homogeneous equality and inequality constraints. Enumerating all extreme rays—given the constraints—is difficult especially if the problem is degenerate and high dimensional. We present and compare two approaches for the parallel enumeration of extreme rays, both based on the double description method. This iterative algorithm has proven efficient especially for degenerate problems, but is difficult to parallelize due to its sequential operation. The first approach parallelizes single iteration steps individually. In the second approach, we introduce a *born/die* matrix to store intermediary results, allowing for parallelization across several iteration steps. We test our multi-core implementations on a 16 core machine using large examples from combinatorics and biology.

Availability: The implementation is available at http://csb.ethz.ch in the tools section (Java binaries); sources are available from the authors upon request.

Keywords: Extreme ray enumeration, metabolic pathway analysis, elementary modes, double description method, born/die algorithm.

1 Introduction

Biochemical reaction networks are typically characterized by a matrix $\boldsymbol{S}^{m\times r}$ containing the stoichometric coefficients, where a negative (positive) entry (i, j) stands for the consumption (production) of metabolite i in reaction j; zero entries indicate that the metabolite does not participate in the reaction. Concentration changes are expressed by

$$\frac{d\boldsymbol{c}}{dt} = \boldsymbol{S}\,\boldsymbol{v} \tag{1}$$

where $\boldsymbol{v}$ represents a flux vector, assigning a flux value to each of the r reactions. For many considerations, (quasi) steady state is assumed, meaning that (internal) metabolite concentrations remain constant. To comply with thermodynamical feasibility, flux values have to be nonnegative (reversible reactions

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 300–309, 2010.

are usually split into a forward and backward part). Hence, flux values are constrained as follows:

$$\boldsymbol{S}\,\boldsymbol{v} = \boldsymbol{0} \tag{2}$$

$$\boldsymbol{v} \geq \boldsymbol{0} \tag{3}$$

The resulting flux space is a (pointed) polyhedral cone. Its extreme rays are called *elementary modes*, and every possible flux vector $\boldsymbol{v}$ can be constructed as nonnegative combination of the elementary modes [1,2].

Enumerating extreme rays is computationally challenging, especially if the problem is degenerate. Polynomial algorithms exist for nondegenerate cases [3], where no point satisfies more than d inequalities for a d-dimensional cone. Unfortunately, many practical problems are degenerate, like most biological examples, and no efficient algorithm is known [4].

Here, we introduce two approaches to parallelize the *double description method* initially invented in [5]. The standard sequential algorithm is described in section 1.1. The parallel approaches are presented in section 2, where we also derive our new *born/die* approach (2.2) and prove the correctness of the algorithm. Finally, we test our multi-core implementation with difficult problems from combinatorics and biology (2.3).

1.1 Double Description Method

The double description method starts with an initial cone and adds remaining constraints iteratively. Each constraint, represented by a halfspace, is intersected with the cone of the previous step (Fig. 1A). The cone is defined by equalities and nonnegativity constraints:

$$P = \{\boldsymbol{x} \in \mathbb{R}^d \mid \boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{0}\ ,\ \boldsymbol{x} \geq \boldsymbol{0}\} \tag{4}$$

This form readily reflects the steady state (eq. 2) and irreversibility constraints (eq. 3) in biochemical reaction networks. Note that any other cone can be transformed into this form.

For the initial cone, we use a special form of the kernel matrix $\boldsymbol{K}$, a basis for the nullspace of $\boldsymbol{A}$. We can assume that $\boldsymbol{A}^{a\times d}$ has full row rank a, and hence we have $d-a$ columns in $\boldsymbol{K}$. Using the row-echelon form for $\boldsymbol{K}$ as proposed in [6], we get:

$$\boldsymbol{K} = [\tilde{\boldsymbol{K}}^T, \boldsymbol{I}]^T \quad \text{with} \quad \tilde{\boldsymbol{K}} \in \mathbb{R}^{a\times(d-a)} \tag{5}$$

Since $\boldsymbol{A}\,\boldsymbol{K} = \boldsymbol{0}$, and since all entries in $\boldsymbol{K}$ are nonnegative for rows $i > a$, the remaining work is to ensure that they are nonnegative for all rows. By iteratively processing the rows $i \leq a$, each column with a negative entry at row i is removed at iteration i. New columns are derived by combining removed columns with other columns with positive entry at row i, such that the value at row i vanishes for the new column.

It is important that columns are only added if they are extreme rays. Thus, every new column is tested for extremity, or equivalently, only adjacent ray pairs

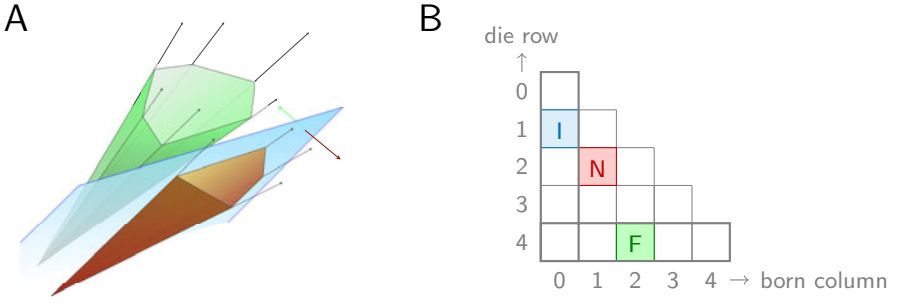


Fig. 1. A: Iteration step of the double description method. The added constraint represented by a halfspace cuts the preliminary cone, possibly generating new extreme rays lying in the seperating hyperplane. **B:** Born/die matrix with sample cells. The highlighted blue cell (I) contains initial rays dying at iteration $t = j + 1$ (here: $t = 2$), that is, rays $\boldsymbol{r}$ in the cell have $r_k \geq 0$ for $k < t$ and $r_t < 0$. The green cell (F) contains final rays, meaning that $r_k \geq 0\ \forall k$. Rays in the red cell (N) are born during the first iteration i, and they die in the third iteration $t = j + 1$.

are used to generate new columns. In fact, most of the computation time is spent for extreme ray or adjacency testing, and we presuppose an efficient method to generate new extreme rays [7,8].

2 Results

2.1 Per-step Parallelization

A first approach to exploit multi-core CPUs parallelizes each iteration step individually. When we intersect the previous cone with the halfspace, new extreme rays are generated and every ray is tested for extremity. This generation and testing process can be executed with multiple threads. We have proposed this technique in [8] and compare it here with an alternative approach applying parallelization across multiple iteration steps.

2.2 The Born/Die Approach

Born/die matrix and basic idea

At iteration t, intermediary extreme rays $\boldsymbol{r}$ with negative value $r_t < 0$ are removed, that is, they *die* at step t. New extreme rays $\boldsymbol{q}$ are created from adjacent ray pairs $(\boldsymbol{r}, \boldsymbol{s})$ with $r_t < 0$ and $s_t > 0$, that is, they are *born* at iteration t. We can thus assign a *born/die* index pair (i, j) to each extreme ray, and for iterations $1, \ldots, n$, we get:

- $i = 0$: initial ray, a column in the initial kernel matrix $\boldsymbol{K}$
- $i \in [1, \ldots, n]$: ray born during iteration $t = i$
- $j \in [0, \ldots, (n-1)]$: ray dies during iteration $t = j + 1$
- $j = n$: ray never dies, final extreme ray, part of algorithm output

Note that $j \geq i$, since a ray can only die *after* being born. Each ray can be associated with a cell in a lower triangular *born/die matrix* (Fig. 1B). The $(n+1)$ columns and rows account for born and die indices, respectively. Column 0 contains initial, row n final extreme rays. The die index j of a ray $\boldsymbol{r}$ is determined as follows:

$$j = \begin{cases} n & \text{if } r_k \geq 0 \quad \forall k \in [1,\, n] \\ \min\{k : r_k < 0,\ k \in [1,\, n]\} - 1 & \text{otherwise} \end{cases} \tag{6}$$

The general idea of the algorithm exploits that every cell in the born/die matrix contains rays dying at a certain iteration step (except for cells in the last row). The matrix is filled up by pairing the dying rays with rays from other cells. The pairing jobs can be run concurrently, but we have to be careful: newly generated rays are again stored in the born/die matrix, and a pairing job should not be started before the involved cells have gathered all rays.

Definition 1. *The generation process of new rays* $\boldsymbol{q}$ *from rays* $\boldsymbol{r}$ *in cell* $(i,\, j)$*, dying at iteration* $j+1$*, and rays* $\boldsymbol{s}$ *adjacent to* $\boldsymbol{r}$ *from a cell* $(k,\, l)$ *with* $s_{j+1} > 0$*, is called* pairing job *or* pairing task $\theta_{(i,\,j)}\Big((k,\, l)\Big)$*. We call the cell* $(i,\, j)$ pairing cell*, and* $(k,\, l)$ partner cell *of* θ*, and we say that pairing cell* $(i,\, j)$ initiates *pairing jobs* $\theta_{(i,\,j)}(\star)$*.*

Cell stages

Before we can use the rays of a cell, we must ensure that the cell has already collected all rays to be stored in it ((A)ccumulating stage). When the cell has collected all rays, it enters the (B)earing stage. After actively triggering pairing jobs, it is still involved in pairing jobs triggered by other bearing cells ((C)ollaborating stage). Finally, if the rays in the cell are not used anymore, they are dropped and the cell is in the (D)one stage. The cell stages (A)-(D) are summarized in the table below, and an exemplarily born/die matrix with cells at different stages is shown in Fig. 2A.

(A)ccumulating: The cell is collecting rays generated by pairing jobs, but is not involved in pairing jobs in a constitutive way.
(B)earing: The cell is initiating pairing jobs as pairing cell. It might also be involved in other pairing jobs as partner cell.
(C)ollaborating: The cell is involved in pairing jobs as partner cell, but has completed the initiation of pairing jobs.
(D)one: The cell is not involved in pairing jobs anymore. If it is not a final row cell, the rays previously stored in the cell have been dropped.

Conditions for cell stage transitions

Lemma 1 (Pairing Lemma). *Let* $t = j+1$*, and* $\boldsymbol{r}$ *be a ray in cell* $(i,\, j)$ *with* $i \in [0,\, n-1]$*,* $j \in [0,\, n-1]$*. Then,* $\boldsymbol{r}$ *is paired with adjacent rays* $\boldsymbol{s}$ *with* $s_t > 0$ *found in cells*

$$\mathit{\Pi}_{(i,\,j)} = \{(k,\, l) : k \in [0,\, j]\ ,\ l \in [j+1,\, n]\}$$

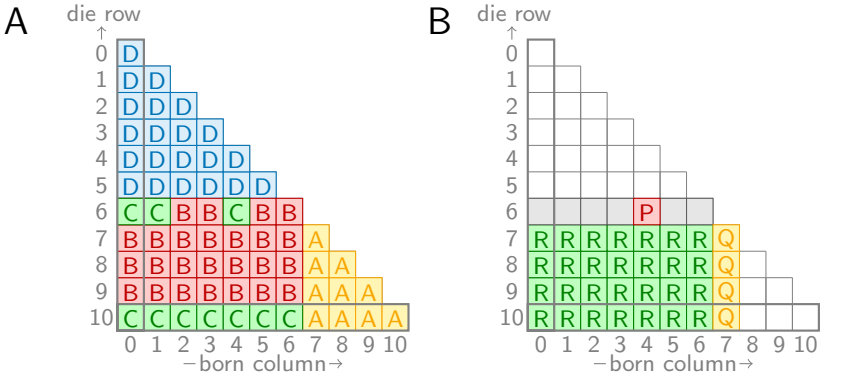


Fig. 2. A: Hypothetical intermediary cell stages (A)ccumulating, (B)earing, (C)ollaborating and (D)one. Collaborating cells can only occur in the final row n, and in the first bearing row ρ (here: $\rho = 6$). Rays in cells of the final row never die, and are thus never bearing; cells in rows between ρ and n have pending partner cells in column $\rho + 1$, which are still accumulating. Cells above row ρ are always in the Done stage, cells with column index $> \rho$ are always accumulating. **B:** Born/die matrix with an exemplarily red pairing cell (P) and yellow destination cells (Q) for pairing jobs involving P. All green cells (R) are partner cells for P. Note that any other pairing cell in row 6 has the same destination and partner cells as P.

Proof. From the main lemma of the double description method (see [9]), we know that at iteration t, rays $\boldsymbol{r}$ with $r_t < 0$ are paired with adjacent rays $\boldsymbol{s}$ with $s_t > 0$. Here, we also have $r_t < 0$ since $\boldsymbol{r}$ is stored in row j, that is, it dies at iteration $t = j + 1$. It remains to prove that partner rays $\boldsymbol{s}$ can only be found in the cells $\mathit{\Pi}_{(i,\,j)} = \{(k,\,l)\}$. Clearly, $k \leq j$, otherwise, the partner ray would not yet be born at iteration t. Furthermore, $\boldsymbol{s}$ must die *after* step t, that is, $l \geq j + 1$. □

Lemma 2 (Dependency Lemma). *Every extreme ray* $\boldsymbol{q}$ *stored in cell* $(i,\,j)$ *with* $i \in [1,\,n]$, $j \in [1,\,n]$ *has been generated from an ancestor ray in*

$$\mathit{\Psi}_{(i,\,j)} = \{(k,\,l) : k \in [0,\,i-1],\; l = i-1\}$$

Proof. Since $i \geq 1$, a ray $\boldsymbol{q}$ stored in cell $(i,\,j)$ must have been generated from a dying ray $\boldsymbol{r}$ with $r_t < 0$ and a surviving ray $\boldsymbol{s}$ with $s_t > 0$. The dying ancestor must be dying during the same iteration step as ray $\boldsymbol{q}$ is born, that is, at iteration $t = i$, and we have $l = i - 1$. It is not relevant when ray $\boldsymbol{s}$ was born, and hence $k \in [0,\,i-1]$. □

The dependency lemma 2 not only tells us when to start pairing jobs, but also where the new rays created from those jobs will be stored. The rays stored in cell $(i,\,j)$ are connected to ancestor cells only through column index i, and all ancestor cells with dying rays are contained in row $i-1$ (Fig. 2B). This leads to the following corollary:

Corollary 1 (Destination Cells). *Rays* $\boldsymbol{q}$ *generated from a pairing job* $\theta_{(\star,\,i)}(\star)$ *initiated by a pairing cell from row* i *are placed in column* $i+1$.

Stages (B) and (C) are *active* since cells are involved in pairing jobs during those stages. The following corollary from lemma 2 yields the condition for the activation of a cell:

Corollary 2 (Activation Condition: transition A–B). *Cells in column* i *with* $i \in [1, n]$ *have collected all rays if all pairing jobs* $\theta_{(\star,\,k)}(\star)$ *involving a pairing cell from row* $k = i-1$ *have completed.*

The sequential algorithm keeps rays which are feasible at least until the next iteration. Infeasible rays are dropped after the generation of new rays with adjacent partner rays. To the same effect, we derive a deallocation condition for rays stored in the born/die matrix:

Proposition 1 (Deallocation Condition: transition C–D). *Rays stored in cells of row* j *with* $j \in [0, n-1]$ *can be dropped if all pairing jobs* $\theta_{(\star,\,k)}(\star)$ *initiated by a pairing cell from row* $k \leq j$ *have completed.*

Proof. As a precondition, the cell in row j is not used as pairing cell, and thus also not as partner cell as implied by pairing lemma 1. Consequently, the rays in the cell are not used in any pairing job, and because $j < n$, they are also not final rays, that is, they can be dropped. □

To account for the last transition from (B) to (C), we count the completed pairing jobs initiated by a cell, and introduce a *remaining job counter* Θ as follows:

$$\begin{aligned}\Theta\Big((i,\,j)\Big) &= \text{remaining jobs, initiated by } (i,\,j)\\ &= |\theta_{(i,\,j)}(\star)|_{\text{total}} - |\theta_{(i,\,j)}(\star)|_{\text{completed}}\\ &= (j+1)(n-j) \;- |\theta_{(i,\,j)}(\star)|_{\text{completed}}\end{aligned} \tag{7}$$

The counter $\Theta\Big((i,\,j)\Big)$ is initialized with $(j+1)(n-j)$ according to pairing lemma 1, and decremented whenever a pairing job $\theta_{(i,\,j)}(\star)$ terminates. We can also derive the activation and deallocation conditions from Θ by computing the minimum index ρ of a row with at least one cell with remaining jobs:

$$\rho = \begin{cases} n & \text{if} \quad \Theta\Big((i,\,j)\Big) = 0 \quad \forall\, i,j \\ \min\{j : \exists i : \Theta\Big((i,\,j)\Big) > 0\} & \text{otherwise} \end{cases} \tag{8}$$

Our observations are summarized in Table 1.

Implementation aspects

For the concurrent execution of pairing jobs, operations modifying the cell counter Θ must be thread-safe. In our Java implementation, we use variables from

Table 1. Conditions for cell stages and algorithm termination

Object	*Stage*	*Condition*
(i, j)	(A)ccumulating	$i > \rho$
(i, j)	active, (B) or (C)	$i \leq \rho \leq j$
(i, j)	(B)earing	(i, j) is active and $\Theta\Big((i, j)\Big) > 0$
(i, j)	(C)ollaborating	(i, j) is active and $\Theta\Big((i, j)\Big) = 0$
(i, j)	(D)one	$j < \rho$
algorithm	terminates	$\rho = n$

the atomic package of the standard library. The classes `AtomicInteger` and `AtomicIntegerArray` offer atomic, thread-safe and lock-free `decrementAndGet` and `compareAndSet` operations. An additional helper variable $\Omega(j)$ for $j \in [0, n-1]$ is initialized with $j + 1$, accounting for the number of cells in row j which are still (or not yet) bearing. The implementation of the conditions in Table 1 is illustrated in pseudo-code procedure `PairingJobTermination`. In addition to the state variables $\Theta\Big((i, j)\Big), \Omega(j)$ and ρ used to control cell stage conditions and algorithm termination, also adding rays to an accumulating matrix cell and job queue operations must be thread-safe.

Procedure `PairingJobTermination`

```
desc: Called before terminating a concurrently executed pairing job θ_(i,j)(⋆)
begin
    jobsLeft ← decrementAndGet Θ((i, j))
    if (jobsLeft = 0) then
        cellsLeft ← decrementAndGet Ω(j)
        while (cellsLeft = 0) do      /* no bearing cells left in row j  */
            if (ρ ← compareAndSet ρ = j, j + 1) then      /* j was ρ  */
                if (ρ = n) then
                    terminate;
                else
                    queue new pairing jobs involving a cell of the activated
                    column ρ
                    j ← j + 1
                    cellsLeft ← Ω(j)
                end
            else
                cellsLeft ← −1            /* j was not ρ ⇒ break loop */
            end
        end
    end
end
```

We have now all elements for the born/die algorithm:

1. Initialize: $\Theta\Big((i,\, j)\Big) = (j+1)(n-j)$ and $\Omega(j) = j+1$
2. Add rays from the initial kernel matrix $\boldsymbol{K}$ to column 0 in the born/die matrix.
3. Setup an empty job queue.
4. Start concurrent working threads, processing queued jobs if available.
5. Activate column 0 by setting $\rho = 0$ and queue new pairing jobs involving only cells from column 0. All further pairing jobs will be added to the queue by terminating jobs as illustrated in procedure `PairingJobTermination`.
6. Await termination condition $\rho = n$.
7. Terminate working threads and return resulting extreme rays stored in matrix row n.

2.3 Experimental Results

We have tested the two approaches on five large problems from combinatorics and systems biology. Two examples are part of the `cddlib` sample files [10]: `ccp7` (116,764 facets of a 0/1 polytope in 21 dimensions) and `mit71-61` (3,149,579 vertices, 60 dimensions). Three examples concern elementary modes (EMs) of central metabolism of *E.coli*, namely `coli-S2` (507,632 EMs corresponding to extreme rays of a polyhedral cone in 64 dimensions, configuration S2 in [8]), `coli-S3` (2,450,787 EMs, 68 dimensions) and `coli-L1` (26,381,168 EMs, 76 dimensions).

The born/die approach seems to have a larger overhead especially with few threads (Table 2), which is not surprising since the work is split into many (possibly fine-grained) pairing jobs. The scale-up behaviour of the methods is comparable, and for some examples, both algorithms scale poorly beyond 8 threads.

Table 2. Computation times for *per-step* and *born/die* approach with 1,2,4,8 and 16 threads (arbitrary precision integer arithmetic for `ccp7` and `mit71-61`, double precision arithmetic for `coli` examples, adjacency test with rank updating and modulo arithmetic), and sequential single-threaded times for `cddlib` (V0.94f with double arithmetic, [10]) and `4ti2` (V1.3.2 with 64 bit integers, [11]). The tests are run on a 64bit linux 2.6.18 machine with 4 Intel Xeon X7350 Quad Core CPUs with 2.93GHz. We used a Sun Java 64-Bit Server VM (1.6.0_15-b03) with at most 8GB memory, except for coli-L1 (32GB). All times (in seconds unless otherwise indicated) are the mean of the median 6 measurements of 10 runs, except for `cddlib` and `4ti2`, where the timing is taken from a single test run. Supplementary information for all tests is available at `http://csb.ethz.ch` in the publications section.

Problem	*Per-step*					*Born/die*					*cddlib*	*4ti2*
Threads	*1*	*2*	*4*	*8*	*16*	*1*	*2*	*4*	*8*	*16*	*(1)*[1]	*(1)*[1]
ccp7	806	418	253	151	97	1,278	626	313	178	138	22,652	525
mit71-61	481	270	158	113	116	539	294	158	116	117	>4d	13,253
coli-S2	38	22	15	14	15	60	29	17	16	14	–	–
coli-S3	338	199	144	121	115	579	319	187	145	138	–	–
coli-L1	5,204	3,229	2,359	1,884	1,925	10,291	5,146	2,989	2,496	2,448	–	–

On the other hand, we have solid speedup for `ccp7`, indicating that scalability depends strongly on the problem instance. Both variants also compete well with the sequential implementations `cddlib` [10] and `4ti2` [11], even if a comparison is difficult[1]. The systems biology examples did not terminate with `cddlib` and `4ti2`, possibly due to missing pre-processing e.g. to compress input matrices [7].

3 Conclusions

We have presented and compared two different approaches to parallelize the inherently sequential double description method. In a per-step approach, we apply parallelization to the individual iteration steps. The second and novel born/die algorithm stores intermediary results in matrix cells, which enables for a cross-step parallelization. Numerous pairing jobs can be run concurrently if certain dependency conditions are considered. Both approaches compete well with alternative implementations if run on a single-core system, and scale comparably well on multi-core architectures. The multi-threaded implementation runs within the same virtual machine. However, we are confident that the born/die approach is also a promising candidate for distributed computation – in particular through its sophisticated storage model for intermediary rays.

Acknowledgment

This work was carried out under the HPC-EUROPA project (contract No. RII3-CT-2003-506079). It was hosted by the Centrum Wiskunde & Informatica (CWI) and supported by SARA Computing and Networking Services, Amsterdam. We thank the scientific hosts, F.J. Bruggeman and L. Stougie.

References

1. Schuster, S., Hilgetag, C.: On elementary flux modes in biochemical reaction systems at steady state. J. Biol. Syst. 2, 165–182 (1994)
2. Stelling, J., Klamt, S., Bettenbrock, K., Schuster, S., Gilles, E.: Metabolic network structure determines key aspects of functionality and regulation. Nature 420, 190–193 (2002)
3. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Appl. Math. 65(1-3), 21–46 (1996)
4. Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., Gurvich, V.: Generating all vertices of a polyhedron is hard. Discrete Comput. Geom. 39(1), 174–190 (2008)
5. Motzkin, T.S., Raiffa, H., Thompson, G., Thrall, R.M.: The double description method. In: Kuhn, H., Tucker, A. (eds.) Contributions to the Theory of Games II. Annals of Math. Studies, vol. 8, pp. 51–73. Princeton University Press, Princeton (1953)
6. Wagner, C.: Nullspace approach to determine the elementary modes of chemical reaction systems. J. Phys. Chem. B 108, 2425–2431 (2004)

[1] Computation times are not directly comparable, because (i) the sequential tools use only one thread, and since (ii) internal pre-processing steps and (iii) input ordering have a significant impact on the running time. In particular (ii) and (iii) affect the order of the constraint processing and hence the number of intermediary rays.

7. Gagneur, J., Klamt, S.: Computation of elementary modes: A unifying framework and the new binary approach. BMC Bioinformatics 5, 175 (2004)
8. Terzer, M., Stelling, J.: Large scale computation of elementary flux modes with bit pattern trees. Bioinformatics 24(19), 2229–2235 (2008)
9. Fukuda, K., Prodon, A.: Double description method revisited. In: Combinatorics and Computer Science, pp. 91–111 (1995)
10. Fukuda, K.: cddlib-094, C–library for polyhedral computations, http://www.ifor.math.ethz.ch/~fukuda/cdd_home/index.html
11. 4ti2 team: 4ti2—a software package for algebraic, geometric and combinatorial problems on linear spaces, www.4ti2.de

Parallelized Transient Elastic Wave Propagation in Orthotropic Structures

Peter Arbenz[1], Jürg Bryner[2], and Christine Tobler[1,2,⋆]

[1] ETH Zürch, Computer Science Department, 8092 Zürich
[2] ETH Zürch, Institute of Mechanical Systems, 8092 Zürich
christine.tobler@sam.math.ethz.ch

Abstract. We discuss the parallelization of a Velocity-Stress FDTD (VS-FDTD) code for the simulation of the propagation of mechanical waves in three-dimensional microstructures. The C++ code has been parallelized using MPI making extensive use of the Blitz++ library for local computation. We present numerical results for pyramid shaped domains, representing the tip of a focusing lens. We discuss the parallel implementation on a cluster consisting of Opteron 250 nodes connected by a high-speed Quadrics QsNet II network.

1 Introduction

The laser acoustic pump-probe technique is well established for non-destructive measurements of thin films and microstructures [11,16,1,15]. An in-depth resolution in the order of 10 nm can easily be achieved with this technique, whereas the lateral resolution is in the order of the wavelength of the laser light, typically 800 nm for near infrared lasers. Microscopic silicon tips can be used as acoustic focusing lenses in order to improve the lateral resolution [9]. The optimization of the geometry of such focusing tips requires the simulation of the wave propagation within the tip. Silicon is a probable candidate for the material of microscopic focusing tips and a widely used material for Micro-Electro-Mechanical Systems (MEMS) components. Single crystal silicon has cubic orthotropic material properties which is the main motivation for the following calculations of the wave propagation in three dimensional anisotropic structures, which are carried out with the VS-FDTD method on a staggered grid.

The Velocity-Stress Finite-Difference Time-Domain (VS-FDTD) method with a staggered grid for mechanical waves was presented 1976 by Madariaga for an axisymmetric two-dimensional model [8]. In 1984 and 1986 Virieux investigated the SH- and P-SV-wave propagation with a staggered two-dimensional VS-FDTD formulation for geophysical problems [13,14]. In 1988 Temple presented a three-dimensional numerical model where the PDEs for the displacement were directly discretized [10]. In this work a three-dimensional model is solved with the VS-FDTD method using the staggered grid that was suggested by Fellinger

⋆ Corresponding author. Present address: ETH Zürch, Seminar for Applied Mathematics, 8092 Zürich.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 310–319, 2010.

et al. [4]. The VS-FDTD approach gives direct access to both, the displacement components and the stress components.

Since the grid spacing is determined by the wave length of the incoming wave, the number of grid points is proportional to the volume of the geometry under consideration. In order to be able to simulate geometries of practically relevant sizes we had to parallelize our VS-FDTD code originally written in MATLAB. We decided to rewrite the code in C++. For optimal portability we chose to parallelize the code using MPI [5,7]. Extensive use of the Blitz++ library should guarantee good local performance [12]. We present this effort with a particular emphasis on the distribution of the data for the pyramid shaped domains that are typical for acoustic focusing lenses. We report on results that we obtained on a Quadrics-connected Beowulf cluster [2].

2 The Problem

The field variables of elastodynamics are the motion-related particle displacement $\mathbf{u}(\mathbf{x},t)$, the velocity $\mathbf{v}(\mathbf{x},t) = \dot{\mathbf{u}}(\mathbf{x},t)$, strain tensor $\boldsymbol{\varepsilon}(\mathbf{x},t)$, and stress tensor $\mathbf{T}(\mathbf{x},t)$. They all are functions of 3D space $\mathbf{x}$ and time t.

If the orthotropic axes of the considered structure are chosen in parallel to the calculation axes then the stress-strain relation is given by

$$\mathbf{T} = \begin{bmatrix} T_{xx} \\ T_{yy} \\ T_{zz} \\ T_{xy} \\ T_{xz} \\ T_{yz} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{12} & 0 & 0 & 0 \\ C_{12} & C_{11} & C_{12} & 0 & 0 & 0 \\ C_{12} & C_{12} & C_{11} & 0 & 0 & 0 \\ 0 & 0 & 0 & 2C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & 2C_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & 2C_{44} \end{bmatrix} \begin{bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{bmatrix} = C\boldsymbol{\varepsilon}. \tag{1}$$

The strains depend on the displacements as

$$\varepsilon_{pq} = \frac{1}{2}(\partial_p u_q + \partial_q u_p), \qquad p = x, y, z, \tag{2}$$

where, for some differentiable function ϕ, $\partial_x \phi = \partial\phi/\partial x$, $\partial_y \phi = \partial\phi/\partial y$, $\partial_z \phi = \partial\phi/\partial z$.

The wave propagation is described by the equations of motion,

$$\rho\, \dot{v}_p = \sum_{q=x,y,z} \partial_q T_{pq} + f_p, \qquad p = x, y, z, \tag{3}$$

by the constitutive equations

$$\begin{aligned} \dot{T}_{pp} &= C_{11}\dot{\varepsilon}_{pp} + C_{12} \sum_{q \neq p} \dot{\varepsilon}_{qq} + g_{pp}, && p = x, y, z, \\ \dot{T}_{pq} &= 2C_{44}\dot{\varepsilon}_{pq}, && pq = xy, xz, yz, \end{aligned} \tag{4}$$

and the kinematic relations,

$$\dot{\varepsilon}_{pq} = \frac{1}{2}(\partial_p v_q + \partial_q v_p), \qquad p, q = x, y, z. \tag{5}$$

Here, the dotted quantities denote time derivatives. In (3), ρ is the material density and f_x, f_y, and f_z are volume force densities. In (4), g_{xx}, g_{yy}, g_{zz}, are stress excitation terms. In our computations we assume zero volume forces.

Substituting the kinematic relations into the constitutive equations leads to the six equations for the normal and shear stresses,

$$\dot{T}_{pp} = C_{11}\partial_p v_p + C_{12}\sum_{q\neq p}\partial_q v_q + g_{pp}, \qquad \dot{T}_{pq} = C_{44}(\partial_p v_q + \partial_q v_p), \tag{6}$$

that together with the equations of motion and appropriate initial conditions for displacements, velocities, and stresses completely describe the wave propagation for the case that the orthotropic axes of the material are in parallel with the coordinate directions x, y, and z.

3 Solution Method

The SV-FDTD discretization is done using a staggered grid as described by Fellinger [4]. In Fig. 1 the positions of the various field components are illustrated. The unit cell with the edge lengths Δx, Δy, and Δz is chosen such that the normal stresses are at the center of the cell, the shear stresses are at the midpoints of the edges, and the velocities are in the middle of the surfaces. Correspondingly, the domain is represented as a structure with $I \times J \times K$ unit cells. I, J, K are the numbers of unit cells in x, y, and z-direction, respectively. The arrangement is designed in a way that spatial derivatives can be replaced by divided differences. The same holds true for the temporal derivatives. We denote

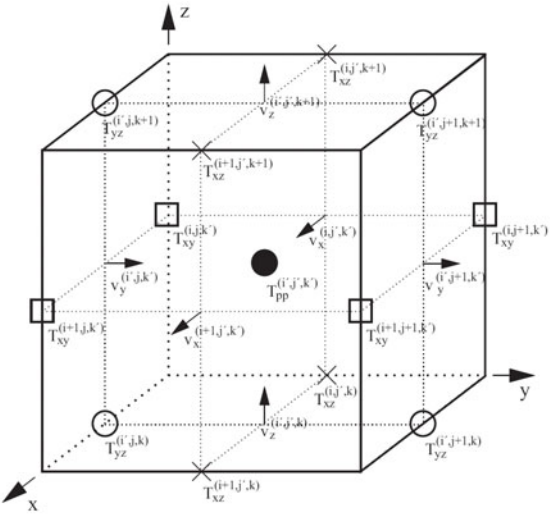


Fig. 1. Unit cell for the discretization at the position x = i, y = j, z = k

a quantity q discretized at $(x, y, z) = (i\Delta x, j\Delta y, k\Delta z)$, $t = \ell\Delta t$ by $q^{(\ell)}_{(i,j,k)}$. Then we advance the discretized velocity field at integer times,

$$v_p^{(\ell)} = v_p^{(\ell-1)} + \dot{v}_p^{(\ell-\frac{1}{2})}\Delta t, \quad p = x, y, z, \tag{7}$$

and the discretized displacement and stress fields at intermediate times,

$$u_p^{(\ell+\frac{1}{2})} = u_p^{(\ell-\frac{1}{2})} + v_p^{(\ell)}\Delta t, \quad T_{pq}^{(\ell+\frac{1}{2})} = T_{pq}^{(\ell-\frac{1}{2})} + \dot{T}_{pq}^{(\ell)}\Delta t, \quad p, q = x, y, z. \tag{8}$$

For the temporal derivative of T_{xx} and T_{xy} we obtain from (6)

$$\dot{T}_{xx(i',j',k')} = C_{11}\Delta_x v_{x(i',j',k')} + C_{12}\sum_{q=y,z}\Delta_q v_{q(i',j',k')} + g_{xx(i',j',k')}, \tag{9}$$

$$\dot{T}_{xy(i,j,k')} = C_{44}\left(\Delta_x v_{y(i,j,k')} + \Delta_y v_{x(i,j,k')}\right), \tag{10}$$

where $i' = i + 1/2$, $j' = j + 1/2$, $k' = k + 1/2$, and the three centered divided differences are given by

$$\Delta_x\phi_{(i,j,k)} = \frac{\phi_{(i+\frac{1}{2},j,k)} - \phi_{(i-\frac{1}{2},j,k)}}{\Delta x}, \quad \Delta_y\phi_{(i,j,k)} = \frac{\phi_{(i,j+\frac{1}{2},k)} - \phi_{(i,j-\frac{1}{2},k)}}{\Delta y},$$

$$\Delta_z\phi_{(i,j,k)} = \frac{\phi_{(i,j,k+\frac{1}{2})} - \phi_{(i,j,k-\frac{1}{2})}}{\Delta z}.$$

Similar equations hold for the rest of the stress components. The discretization of, e.g., the first equation of motion (3) reads

$$\rho\,\dot{v}_{p(i,j',k')} = \sum_{q=x,y,z}\Delta_q T_{pq(i,j',k')} + f_{p(i,j',k')}. \tag{11}$$

As the *initial conditions* the displacements $\mathbf{u}^{(0)}_{(i,j,k)}$, velocities $\mathbf{v}^{(0)}_{(i,j,k)}$, and stresses $\mathbf{T}^{(0)}_{(i,j,k)}$ must be prescribed.

All simulations presented below are calculated with *stress-free boundary conditions*. The shear stress grid points positioned at the boundaries are simply set to zero. The velocity grid points at the boundaries are calculated by one-sided differences assuming that the normal stresses at the boundaries vanish, see Bryner *et al.* [3] for details.

A *stability condition* limits the time step [4,3],

$$\Delta t \le \frac{\Delta x}{\sqrt{3}c_{po}},$$

where c_{po} is the speed of the fastest primary wave.

4 Implementation Details

We implemented the VS-FDTD algorithm in C++ for parallel processing, using the Blitz++ library [12] for storing data and MPI for passing messages [5,7].

In this paper we consider domains of the shape of a pyramid with square base. We store the variables T_{pq} and u_p, $p, q = x, y, z$, layer by layer from the base ($z = 0$) to the tip of the pyramid. Because of the symmetry in the geometry, only a quarter of the data has to be stored. On the symmetry planes $x = 0$ and $y = 0$ homogeneous Neumann boundary conditions are imposed. In each time step, updating the variables T_{pq} and u_p requires individual `for`-loops for interior points, boundary points, and points on the symmetry planes.

For the parallelization, the pyramid is partitioned in z-direction, from base to tip. Each processor is given an equal fraction of the volume of the pyramid. So, the number of layers owned by a processor increases from base to tip, cf. the sketch in Fig. 2. Logically, the processors are arranged in an array such that each of them has at most two neighbors. In each iteration step of the FDTD algorithm, information on interface layers is exchanged among neighboring processors.

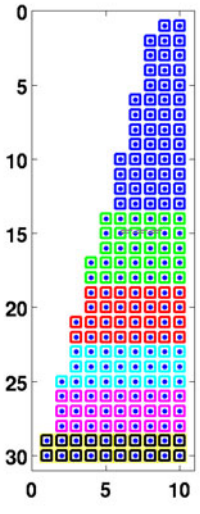


Fig. 2. Pyramid distributed on 6 processors. View on a symmetry plane.

In a first approach, the variables were stored in a 3D-array each. Storage for data at grid points outside of the pyramid was allocated but never accessed. The data was distributed according to the workload which is proportional to the grid points inside the pyramid. So, the processors handling the top portions of the pyramid store many more data layers than the others. This storage scheme was easy to implement but entailed a severe restriction on the problem sizes we could deal with.

Therefore, in a second approach, each of the 3D-arrays was replaced by an array of 2D-arrays where each of the latter stores quantities of a horizontal layer of the pyramid. The number of elements per layer and thus per 2D-array decreases from the base to the tip. In this approach, the data *and* computational load are balanced equally among the processors. However, the volume of the data that has to be communicated among processors is much smaller close to the tip of the pyramid than at the base.

Regarding communication, the straightforward strategy is to exchange the data corresponding to the layers interfacing two processors. This requires communicating 9 messages in each direction, the six stresses T_{pq} and the three velocities v_p. Some of the communication can be hidden by computation. A second strategy is to communicate the velocity data v_x, v_y and v_z corresponding to *two* layers nearest to the interface. In this case, only three (however double-sized) messages need to be sent per time step. The missing T_{pq} values can be recovered

by means of formula (9) and (10). In this way we reduce the communication volume by one third, at the expense of calculating some values twice.

For the run-time analysis let us assume that the base layer contains $m \times m$ grid points and that there are n layers from base to tip of the pyramid. If the top layer consists of just one grid point then there are about $m^2 n/3$ grid points inside the pyramid. We distribute the data on the processors in contiguous layers such that each of the processors handles about $m^2 n/3p$ grid points. The volume of the messages that are exchanged between processors j and $j+1$ is not easily estimated. The largest messages are about m^2 data items long; the smallest messages are considerably smaller. Since there are about 75 floating point operations per grid point and time step, the overall execution time for the first approach on p processors is about

$$t_p \approx \left(\frac{75 m^2 n}{3p} + 4 \cdot 30 m^2 \right) t_{\text{flop}} + 6\, t_{\text{startup}} + 6\, m^2\, t_{\text{word}}. \tag{12}$$

Here, t_{flop}, t_{startup}, and t_{word} denote the times for executing one (double precision) floating point operation, for the communication startup overhead (MPI latency) and for sending one (8-byte) word of data [6]. We discuss these numbers in the next section.

According to Amdahl's law [6], a parallel program executes efficiently only if the fraction that does not scale well is relatively small. Here, besides the communication, this fraction consists of the redundant work in the two double-layers at the interfaces of the subgrids.

5 Numerical Results

The calculations were carried out on Brutus, a Beowulf cluster at ETH Zurich [2]. The subcluster we used consists of compute nodes, each one containing two AMD Opteron 250 processors running at 2.4 GHz and 8 GB of RAM. The nodes are interconnected via a high-speed Quadrics QsNet II network of about 900 MB/s band width and 1.25 μs MPI latency. So, for the quantities in (12) we get $t_{\text{startup}} = 1250$ ns and $t_{\text{word}} = 8.88$ ns. From one-processor runs we estimate $t_{\text{flop}} \approx 1$ ns. This also corresponds roughly to the memory bandwidth of the Opteron. Note that t_{flop} very much depends on the kind of computation that are performed.

The first simulation is of a pyramid with an opening angle of 13.7°, a quadratic base of 1.5 μm side length and a tip of 0.1 μm side length. The width of the incoming wave is 1.4 μm, fading out to the sides with a Hanning distribution. The computational domain is embedded in a $377 \times 377 \times 1442$ grid, corresponding to 6.5 GB of memory for the second approach. The layer at the tip consists of 26×26 grid points. We simulated a period of 700 ps, using 7000 time steps.

The second simulation pyramid has an opening angle of 20°, 1.7 μm wide at the base and 0.1 μm at the tip. The incoming wave is the same as with the first simulation. The size of the rectangular grid encasing the computational domain is $427 \times 427 \times 1102$ grid points, corresponding to 6.4 GB of memory. The tip

layer again consists of 26^2 grid points. We simulated a period of 530 ps, using 5300 time steps.

Both simulations were executed on 16 and 64 processors (i.e., 8 and 32 nodes), using the first and second approaches described in Section 4. (They are too big to be executed on a single processor.) The execution times and speedups are listed in Table 1.

Table 1. Execution times in seconds for the two simulations (using 16 processors)

Opening angle	Number of processors	First approach time	First approach speedup	Second approach time	Second approach speedup
13.7°	16	15730 s		12244 s	
	64	3789 s	4.15	3582 s	3.42
20°	16	10547 s		7563 s	
	64	2753 s	3.83	2485 s	3.04

Comparing the first to the second approach, the memory usage is reduced considerably, by about a factor of 3. The execution time is not reduced so much of course, but only by roughly 25% using 16 processors, and by roughly 10% using 64 processors. The latter is due to the fact that the load could be balanced reasonably well on 64 processors as the tip portion did fit in the memory of a single processor.

Comparing execution times of the 16 to 64 processor runs, we observe speedups between 3.04 and 4.15, corresponding to a relative efficiency between 75% and over 100%. The first approach scales better than the second approach. This is because the processors that handle the grid close to the tip cannot store as many layers as load balance would require. This situation is improved for 64 processors. Since the first (13.7°) model has more layers than the second (20°) model (1441 vs. 1102) the overhead due to redundant computation (2nd term in (12)) is relatively smaller. So, we observe better speedups with the former.

The one-dimensional block data distribution takes its toll. As just noticed, if the number of layers is not big enough then the nonscaling portions in (12), communication and redundant computation, soon become significant.

The results that interest the engineers, the wave propagation in the orthotropic structure, are given in Fig. 3 to 5 for the pyramid with an opening angle of 13.7°. As displayed in Fig. 3, after 100 ps there are two main wave packets moving down the pyramid. The wave packet in front consists of quasi-longitudinal waves (vertical components), the second wave packet of quasi-shear waves (horizontal components). From analytical results, the velocities of these wave packets should be 8432 m/s and 5843 m/s, respectively. This agrees with the results of the simulation. Fig. 4 shows the situation after 300 ps: due to reflection at the borders of the cylinder, there are two additional wavefronts, with other minor ones. After 390 ps, the waves have been reflected at the tip of the pyramid, see Fig. 5.

Finally, we discuss another speedup analysis that we conducted with a smaller model that could be run on a single processor. This example has a base of

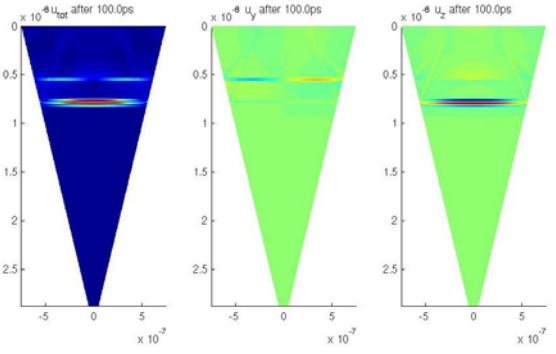


Fig. 3. The displacement fields on the yz-symmetry plane after 100 ps. u_{tot}: displacement amplitude, u_y: displacement in y direction, u_z: displacement in z direction.

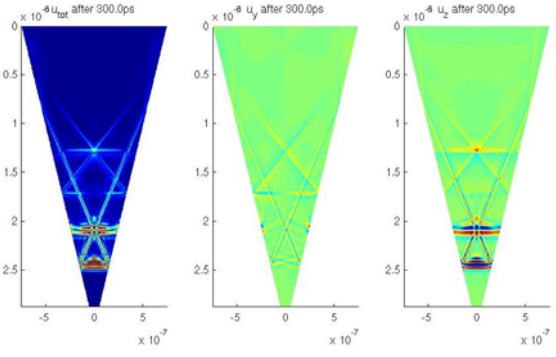


Fig. 4. The displacement fields on the yz-symmetry plane after 300 ps. u_{tot}: displacement amplitude, u_y: displacement in y direction, u_z: displacement in z direction.

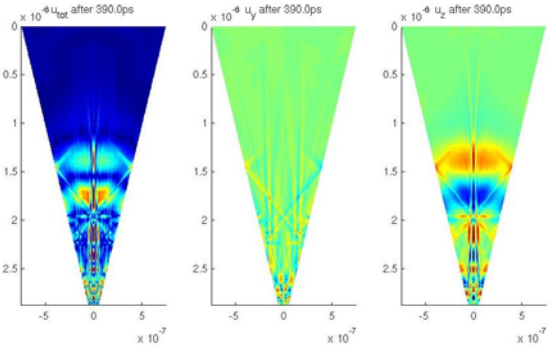


Fig. 5. The displacement fields on the yz-symmetry plane after 390 ps. u_{tot}: displacement amplitude, u_y: displacement in y direction, u_z: displacement in z direction.

Table 2. Analysis of the $176 \times 176 \times 618$ case

Number of processors	1	2	4	8	12	16
Execution time	1574 s	834 s	448 s	232 s	162 s	133 s
Speedup	1.0	1.9	3.5	6.8	9.7	11.8
Efficiency	1.0	0.94	0.88	0.85	0.81	0.74
Formula (12)	1	0.99	0.97	0.93	0.89	0.85

$0.696\,\mu$m, an opening angle of 13.7° and a simulation time of 80 ps, corresponding to 800 time steps. The size of the computational domain is $176 \times 176 \times 618$, which corresponds to 678 MB of memory. Only the second approach is analyzed.

The simplistic formula (12) reproduces reasonably well the efficiencies given in Tab. 2. It however does not take into account that the nodes have two processors that communicate through a shared memory with insufficient memory bandwidth.

These data again confirm that the redundant portion of the computation which is a fixed amount of work independent of the processor number p dominates very soon, already for small processor numbers. This is not too surprising for the factor m^2, i.e., the number of grid points in the base layer, and the fact that the processors close to the base store only very few layers, cf. Fig. 2.

A last simulation concerns weak scalability. The small model of Table 2 was solved with a higher resolution: the cell sizes Δx, Δy, Δz were halved, as was the time step Δt. As a result, the memory consumption and computational cost increases by a factor of 8 and 16, respectively. This simulation was executed on 128 processors. For a perfectly scaling program, the execution time would correspond to twice that of 16 processors in the original model in Table 2, i.e., $2 \cdot 133$ s. The actual execution time was 1039 s which is 3.9 times as much. Thus, the relative efficiency is 26%. The reason for the low efficiency is, again, that, after the lowest slices have reached their minimal width, adding more processors does not decrease the overall execution time any further.

6 Conclusions

The parallelization of our VS-FDTD code was successful in that we can now solve the problem of the relevant sizes. The problem sizes are determined by the geometry of the orthotropic structure to be analyzed and by the wave length of the incoming wave.

The speedups that we measured with a small test example on one to 16 processors of the Beowulf cluster at ETH Zurich were satisfactory. This cluster connects dual-core Opteron 250 nodes by a Quadrics QsNet II network. The solution of relevant problem sizes requires 1–4 hours wall clock time on 16–64 processors.

The parallelization strategy is based on an array arrangement of the processors that is quite easy to adapt to the pyramidal geometry of our structures. In this way it is easy to balance the computational work. However, the communication volume among adjacent processors is varying considerably. The 1D

arrangement of the processor becomes the bottleneck of the computations if too many processors are employed for solving a problem.

References

1. Bonello, B., Perrin, B., Romatet, E., Jeannet, J.C.: Application of the picosecond ultrasonic technique to the study of elastic and time-resolved thermal properties of materials. Ultrasonics 35(3), 223–231 (1997)
2. Beowulf cluster Brutus website: http://www.clusterwiki.ethz.ch/brutus
3. Bryner, J., Vollmann, J., Aebi, L., Dual, J.: Wave propagation in pyramidal tip-like structures with cubic material properties. Wave Motion (2009), doi:10.1016/j.wavemoti.2009.07.003
4. Fellinger, P., Marklein, R., Langenberg, K.J., Klaholz, S.: Numerical modeling of elastic-wave propagation and scattering with EFIT – elastodynamic finite integration technique. Wave Motion 21(1), 47–66 (1995)
5. Gropp, W., Lusk, E., Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-passing Interface, 2nd edn. MIT Press, Cambridge (1999)
6. Kumar, V., Grama, A., Gupta, A., Karypis, G.: Introduction to Parallel Computing. Benjamin/Cummings, Redwood City (1994)
7. Open MPI website: http://www.open-mpi.org
8. Madariaga, R.: Dynamics of an expanding circular fault. Seism. Soc. Am. 66(3), 639–666 (1976)
9. Profunser, D.M., Vollmann, J., Dual, J.: Ultrasonic wave propagation in focussing tips with arbitrary geometries. Ultrasonics 40(1), 747–752 (2002)
10. Temple, J.A.G.: Modeling the propagation and scattering of elastic-waves in inhomogeneous anisotropic media. J. Phys. Appl. Phys. 21(6), 859–874 (1988)
11. Thomsen, C., Grahn, H.T., Maris, H.J., Tauc, J.: Surface generation and detection of phonons by picosecond light-pulses. Phys. Rev. B 34(6), 4129–4138 (1986)
12. Veldhuizen, T.: Blitz++ User's Guide (March 2006), http://www.oonumerics.org/blitz/
13. Virieux, J.: SH-wave propagation in heterogeneous media: velocity-stress finite-difference method. Explor. Geophys. 15(4), 265 (1984)
14. Virieux, J.: P-SV wave propagation in heterogeneous media: velocity-stress finite-difference method. Geophysics 51(4), 889–901 (1986)
15. Vollmann, J., Profunser, D.M., Dual, J.: Sensitivity improvement of a pump-probe set-up for thin film and microstructure metrology. Ultrasonics 40(1), 757–763 (2002)
16. Wright, O.B.: Laser picosecond acoustics in double-layer transparent films. Optics Letters 20(6), 632–634 (1995)

Parallel Numerical Solver for Modelling of Electromagnetic Properties of Thin Conductive Layers

Raimondas Čiegis[1], Žilvinas Kancleris[2], and Gediminas Šlekas[2]

[1] Vilnius Gediminas Technical University
Saulėtekio al. 11, LT10223 Vilnius, Lithuania
rc@fm.vgtu.lt
[2] Semiconductor Physics Institute
A. Goštauto 11, LT01108 Vilnius, Lithuania
kancleris@pfi.lt

Abstract. In this paper we develop the parallel numerical algorithm for modelling of electromagnetic properties of thin conductive layers. The explicit finite difference scheme is obtained after approximation of the system of differential equations on the staggered grid. The parallelization of the discrete algorithm is based on the domain decomposition method. The tool of parallel linear algebra objects ParSol is used to get a semiautomatical implementation of this parallel algorithm on distributed memory computers. Some results of numerical simulations are presented and the efficiency of the proposed parallel algorithm is investigated.

Keywords: parallel algorithms, finite-difference method, numerical simulation, semiconductors, wave radiation.

1 Introduction

Thin high conductivity layers, in particular the layers with two-dimensional electron gas, are widely employed in high frequency electronics. Microwave detectors are not the exception. In [3] the planar asymmetrically shaped homogeneous semiconductor structure for the measurement of microwave power has been proposed. Due to asymmetrical geometry of the structure the non-uniform electron heating takes place in it when the detector is subjected by the microwaves and a DC voltage appears on the ends of the structure enabling microwave power measurement. Since electron heating effect is put on a basis of the proposed device performance, it can be used in a wide frequency range up to a terahertz region [11,12], where a traditional semiconductor diode based on microwave current rectification is of a little use. To measure microwave power the proposed detector either was inserted into waveguide [3,11] or the measurements in free space were performed by direct illuminating of the sensor by terahertz radiation [11,12]. In the latter case the metallic parts of the structure act as a bow-tie antenna coupling the structure with terahertz radiation. Although the detector

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 320–329, 2010.

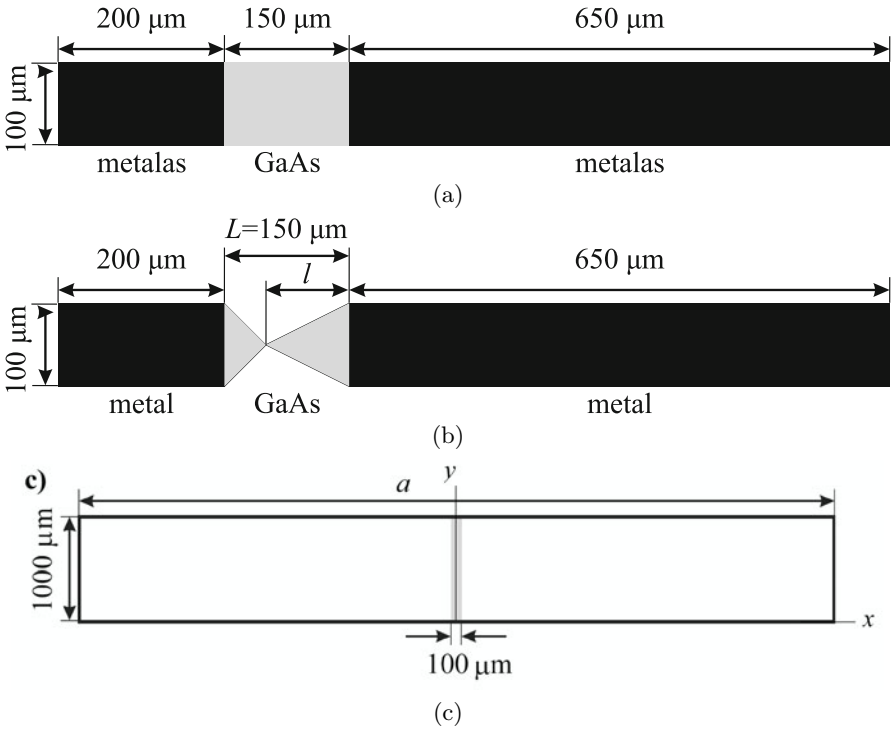


Fig. 1. The structures under investigation: (a) the simplified model, (b) a nicked mesa, (c) their installation into narrowed waveguide. Grey colour in (a) and (b) shows semiconducting layer – mesa, black colour corresponds to the metallic contacts.

is in general a multilayered structure grown on a semi insulating substrate, the thin layer conducting current through the structure is of a great importance for the whole device performance. Thus interaction of electromagnetic wave with the conductive layer should be considered and the electric field distribution in it should be calculated to determine the characteristics of the proposed sensor or engineer the sensor with desirable characteristics.

In the present paper we have concentrated on the interaction of an electromagnetic wave with a thin conducting layer placed in the waveguide. Millimeter wave frequency region has been considered. We have used finite-difference (FD) method originally proposed by Yee [8] and described in a monograph [13]. Since the thickness of the active layer is much less than the wavelength of the electromagnetic field, direct account of it in the FD calculation procedure needs very fine grid. Due to complicated geometrical shape of the structure fine grid in the transverse direction is also desirable. To account the thin material layer in the FD calculation we use the method proposed by Maloney and Smith [10]. This approach allows to use space step size that is much larger than the thickness of the layer.

2 Mathematical Model

The shape of the considered structures is shown in Figure 1. In Figure 1(a) the simplified model of the structure is shown. It consists of the thin layer of GaAs inserted between two metallic layers covering its ends. The length of the mesa (i.e. semiconductor layer) comprises roughly 1/7 part of the whole length of the structure. A nicked mesa is depicted in Figure 1(b). It is seen that at $l/L = 0.5$ the structure is symmetrical, the asymmetry of the mesa depends on the ratio l/L. The electric field in the nicked mesa becomes inhomogeneous when the structure is subjected by the microwave radiation. Overall length of the structure was 1 mm, therefore it is placed in the waveguide in the plain normal to the falling electromagnetic wave and a vector of electric field is oriented along the sample. The considered sample occupies full height of the narrowed waveguide window. It is schematically shown in Figure 1(c).

Depending on the considered frequency range the size of the wide wall a is changed. We have considered two frequency bands: Ka-band, $a = 7.2$ mm, $f = 26-40$ GHz and U-band, $a = 4.4$ mm, $f = 43-65$ GHz. Wishing to check up the algorithms used for calculation we also considered the solid semiconducting layer occupying full window of the waveguide. In this case computed results have been compared with an analytical solution. Investigated mesas are characterized by their width w, thickness d, a specific conductivity σ and a relative dielectric constant ε. The ratio l/L was the characteristic of the nicked mesas. In the waveguide a regular TE_{10} (H_{10}) type wave has been excited. The electric field distribution in the mesa, its average value and the reflection coefficient from the structure were the main characteristics of the model considered in the paper.

FD method for the calculation of electromagnetic field components was used [13]. We employed a Cartesian coordinate system and dimensionless coordinates and time: x/a, y/a, z/a, tv/a where v is the velocity of light in free space and a is a size of wide wall of the waveguide. The section of the waveguide has been simulated the length of which corresponds roughly to two wavelengths of excited oscillations in a waveguide λ_b. At some distance from the beginning of the section the TE_{01} type wave is excited. It propagates into both sides from the excitation plane. The investigated structure is placed at one wavelength in the waveguide ahead from the excitation plane and at the same distance before the end of the modelled waveguide section. Non reflecting boundary conditions have been fulfilled on both open ends of the waveguide section.

Before the obstacle the partly standing wave is formed from the amplitude of which the reflection coefficient from the investigated structure has been determined. As it is seen from Figure 1(c) the investigated layer is inserted in the center of the waveguide window. Therefore, saving computer resources only half of the window has been modelled.

Note, that in a vicinity of the inhomogeneous structure all electromagnetic field components might appear. Therefore, to determine the electric field components in the layer, the Maxwell equations have to be solved computing all six components of the electric and magnetic fields. Making use of dimensionless variables and measuring up the magnetic field strength in electric field units $Z_0 H$,

where Z_0 is an impedance of free space, the well-known Maxwell equations in the 3D semiconductor plate can be written in a following way [7]:

$$\frac{\partial \boldsymbol{E}}{\partial t} = \frac{1}{\varepsilon}(\nabla \times \boldsymbol{H} - \gamma \boldsymbol{E}), \quad \frac{\partial \boldsymbol{H}}{\partial t} = -\nabla \times \boldsymbol{E}. \tag{1}$$

Here $\gamma = Z_0 a \sigma$ accounts for losses in the structure. It was assumed that $\mu = 1$ for the entire simulation area. Outside the semiconductor obstacle $\gamma = 0$ and $\sigma = 1$. The metal surfaces of the structure have been treated as a perfect metal conductor. Calculated electric field has been normalized to the maximum of the electric field component E_y in the empty waveguide.

3 Finite Difference Scheme

The differential problem is approximated by using the finite difference method which is applied on the staggered grids in space and time [13]. The grid, where the particular component is computed is shifted by a half of the step with respect to the other components (see [8], where this procedure was proposed). Let consider the reference grids in space

$$\omega_h = \{(x_i, y_j, z_k) : \ x_i = ih_x, \ y_j = jh_y, \ z_k = kh_z, \ 0 \le i, j, k \le N_w, \ w = i, j, k\}.$$

and time $\omega_t = \{t^n : t^n = nh_t, \ n = 0, \ldots, N_t\}$. As example, let us consider the equation for the component E_x defined in a free space:

$$\varepsilon \frac{\partial E_x}{\partial t} = -\frac{\partial H_y}{\partial z} + \frac{\partial H_z}{\partial y}.$$

Then we define functions E_x, H_y and H_z on grids shifted with respect to each other in space and time coordinates: $E^n_{x,i,j-0.5,k+0.5}$, $H^{n+0.5}_{y,i,j-0.5,k}$, $H^{n+0.5}_{z,i,j,k+0.5}$. All derivatives are approximated by using the forward or backward finite differences, giving the symmetrical approximation formulas:

$$E^{n+1}_{x,i,j-0.5,k+0.5} = E^n_{x,i,j-0.5,k+0.5} - \frac{h_t}{\varepsilon h_z}\left(H^{n+0.5}_{y,i,j-0.5,k+1} - H^{n+0.5}_{y,i,j-0.5,k}\right) \tag{2}$$

$$+ \frac{h_t}{\varepsilon h_y}\left(H^{n+0.5}_{z,i,j,k+0.5} - H^{n+0.5}_{z,i,j-1,k+0.5}\right).$$

Due to the fact that electric and magnetic fields are calculated at different time moments and on staggered grids in space, we get the second order accuracy of approximation of space and time derivatives. We note that a similar strategy was used also to solve numerically systems of nonlinear Schrödinger type equations arising in simulation of laser devices [5,9].

The implementation of the discrete scheme is very simple and efficient, since the discrete solution is obtained by a direct explicit algorithm. The main limitation of this finite difference scheme consists in its conditional stability, which gives a CFL type restriction on the discrete time step.

Our main goal is to simulate devices where a thin layer of GaAs is inserted between two metallic layers. Since the proposed finite difference scheme is only conditionally stable, we would like to avoid the usage of non-uniform (or adaptive) space grids, since small space steps in some part of domain will lead to necessity to select a very small time step h_t. We assume the thin semi-conductor layer has a small thickness in the x direction and is large in the y and z directions.

A subcell method is proposed in [10] for including thin material layers in the finite-difference method. It introduces additional cells where the electric field component normal to the layer is split into two parts. One of the split components accounts for the properties of the layer, while in the other one the electric field component in the free space close to the layer is stored. In the case of E_x component the solution in the free space $e^{n+1}_{xo,i,j-0.5,k+0.5}$ is calculated by using the standard approximation (2). However the electric field component in the thin metal layer is computed by using the following modified equation

$$\varepsilon_s \frac{e^{n+1}_{xs,i,j-0.5,k+0.5} - e^{n}_{xs,i,j-0.5,k+0.5}}{h_t} + \gamma \frac{e^{n+1}_{xs,i,j-0.5,k+0.5} + e^{n}_{xs,i,j-0.5,k+0.5}}{2} \tag{3}$$
$$= -\frac{h^{n+0.5}_{y,i,j-0.5,k+1} - h^{n+0.5}_{y,i,j-0.5,k}}{h_z} + \frac{h^{n+0.5}_{z,i,j,k+0.5} - h^{n+0.5}_{z,i,j-1,k+0.5}}{h_y}.$$

The accuracy of approximation of this component reduces to the first order. The averaged value of both components e_{xo} and e_{xs} have to be taken into account when magnetic field components tangential to the layer are calculated. The details of the application of this technique for the calculation of semiconductor obstacle placed in the waveguide can be found in [7].

4 Parallel Numerical Algorithm

The parallel algorithm is based on the domain decomposition method. The discrete grid w_h is distributed among p processors. The load balancing problem is solved at this step: each processor obtains the same number of grid points and the sizes of overlapping regions of subdomains are minimized (due to the stencil of the grid, processors require information from the neighbouring processors). In this paper we apply the 3D block domain decomposition algorithm, decomposing the grid in all three directions. Since the proposed finite difference scheme is explicit, the parallel version of the solver implements exactly the sequential version of the algorithm. The parallel algorithm is generated semiautomatically by using the tool of parallel linear algebra objects ParSol [6] (for more applications of ParSol see [4,5]). Parallel vectors, which are used to store the discrete solution, are created by specifying three main attributes:

- the dimension of the parallel vector is 3D;
- the topology of processors is 3D;
- the 3D grid stencil is defined by the 27 points box.

Next we present some results of computational experiments. Computations were performed on Vilkas cluster of computers at Vilnius Gediminas Technical University, consisting of nodes with Intel(R) Core(TM)2 Quad processor Q6600. Four processing cores are running at 2.4 GHz each and sharing 8 MB of L2 cache and a 1066 MHz Front Side Bus. Each of the four cores can complete up to four full instructions simultaneously. Obtained performance results are presented in Table 1. Here for each number of processors p the coefficients of the algorithmic speed up $S_p = T_1/T_p$ and efficiency $E_p = S_p/p$ coefficients are presented. $T_p(N_x, N_y, N_z)$ denotes the CPU time required to solve the problem on the given grid ω_h using p processors. We have solved the discrete problem on two grids of sizes $(131 \times 51 \times 284)$ and $(261 \times 101 \times 569)$.

Table 1. Results of computational experiments on Vilkas cluster

	2×1	1×2	4×1	2×2	1×4	8×1	4×2	16×1	8×2
$S_p(1)$	1.98	1.63	3.27	3.11	1.65	6.14	5.59	11.84	10.8
$E_p(1)$	0.99	0.82	0.82	0.79	0.42	0.77	0.70	0.74	0.68
$S_p(2)$	1.98	1.63	3.27	3.11	1.65	6.14	5.59	11.84	10.8
$E_p(2)$	0.99	0.82	0.82	0.79	0.42	0.77	0.70	0.74	0.68

The presented results confirm the conclusions of the standard scaling analysis of the parallel DD algorithm. In the case when up to two cores per node are used, the efficiency is improved, when the size of the problem is increased. It can be recommended to use till two cores per one node in practical simulations. In the case of four cores per node data reading/writing operations start to be a bottle-neck of the parallel algorithm, since the discrete scheme is defined on a very sparse stencil and no intensive local computations are needed to update the solution at each grid point.

5 Simulation Results and Discussions

Initially we compute the reflection coefficient from the conducting layer that fills whole waveguide's window. On the one hand, such installation of the layer simplifies numerical computation since problem reduces to a two-dimensional and subcell method [10] might be compared with a standard FD method in a wider range of the layer thickness. On the other hand, the reflection coefficient from the layer can be found analytically therefore one can estimate the accuracy of numerical calculation. Results of calculations have shown that the standard FD method and the Maloney-Smith method lead to very similar solutions. But in the subcell method the step in a wave propagation direction $dz = 50\mu m$ has been used and the thickness of the layer thickness is roughly four orders less than the step.

Considering the structures we confine ourselves to a fixed thickness of the mesa $d = 0.1\mu m$ and width $w = 100\mu m$ those correspond to the experimentally

realized structures [2]. Initially we investigated the simplified structure shown in Figure 1(a) having different specific conductivities of the mesa. Calculated dependencies of the reflection coefficient and average electric field component E_y on mesa's specific electrical conductivity are shown in Figure 2.

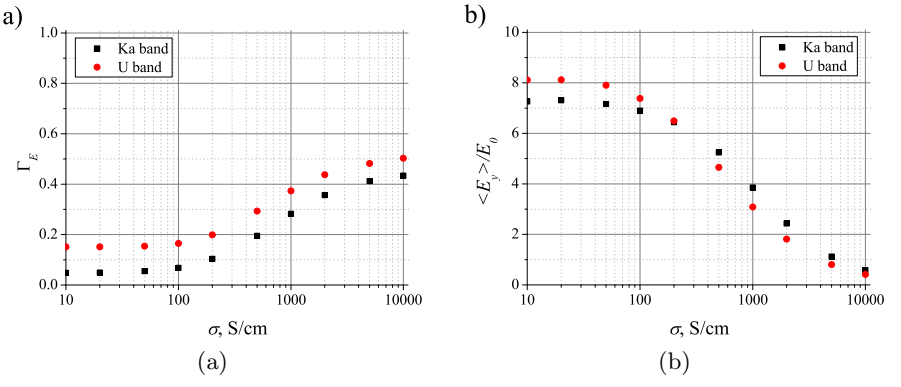


Fig. 2. The dependence on a conductivity of the mesa for different frequency bands: (a) of the reflection coefficient for the structure with uniform mesa, (b) of the average electric field in it. Calculation parameters: $\varepsilon = 12$, $d = 0.1\mu m$, $f = 33.4$ GHz for Ka-band and $f = 54.3$ GHz for U-band.

As one can see from the figure, when the conductivity of the mesa decreases the reflection coefficient and average electric field are approaching some limiting values defined by the properties of the metallic layers rather than by the properties of the mesa. Since metallic layers comprise significant part of the investigated structure they caused in themselves some reflection. Reflection coefficient is larger for U-band since the normalized width of the structure w/a is larger in comparison with the same quantity for Ka-band. The increase of σ results in the growth of the reflection coefficient, whereas the averaged electric field in the mesa decreases. At $\sigma = 0$, the equivalent circuit of the structure is comprised of two inductances with a capacitance in between. While the specific conductivity of the mesa increases conductive currents begin to shunt the capacitance causing in turn the increase of the reflection since the conductive current begins to dominate in a whole structure under consideration.

For the investigation of nicked mesa we choose the specific conductivity of it 500 S/cm. At such conductivity the electric field distribution within uniform mesa is homogeneous for both frequency bands under consideration and the averaged electric field in it remains sufficiently large. Calculation results for the nicked mesa with the different ratios of l/L are shown by points in Figure 3. Minimum width of the mesa corresponds to the size of the step in the x direction $dx = 5\mu m$.

As one can see from the figure the distribution of the electric field is symmetrical for the symmetrical mesa ($l/L = 0.5$) and asymmetry is growing with the

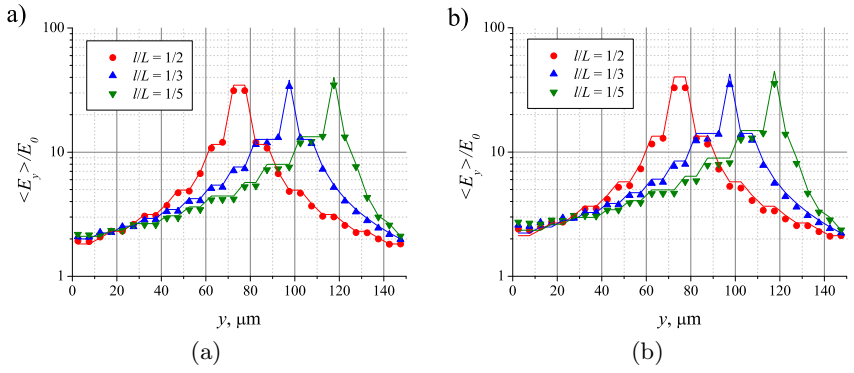


Fig. 3. Dependence of the component E_y of electric field averaged in x direction on a coordinate y for the nicked mesa with different ratio l/L: (a) $f = 33.4$ GHz, Ka-band, (b) $f = 54.3$ GHz, U-band. Here $y = 0$ corresponds to the left metal-mesa interface in Figure 1(b). Points correspond to the calculation results, solid line demonstrates approximation (4). Calculation parameters: $\varepsilon = 12$, $\sigma = 500$ S/cm, $d = 0.1\mu m$, the ratio l/L is indicated in the figure.

asymmetry of the mesa. The maximum of the electric field appears in the narrowest part of the mesa. It should be noticed that it is practically independent of the ratio of l/L. Furthermore, comparing results shown in Figure 3(a) and (b) it is seen that for investigated mesas electric field distribution is similar in both frequency bands. Distribution of electric field is not sensitive to the frequency within considered frequency band as well.

These facts supports the idea that the electric field in the nicked conductive mesa is practically determined by its shape and the dependence of the electric field on coordinate can be approximated by the expression obtained by solving Poisson equation in a DC electric field for a special case when the drift velocity of the electrons is independent of the electric field. For this simple approach electric field amplitude in the mesa can be expressed as [1]

$$\langle E_y \rangle(y) = \langle E_y \rangle_c \frac{w_c}{w(y)}, \tag{4}$$

where $\langle E_y \rangle_c$ is the electric field amplitude near the mesa-metal interface, w_c and $w(y)$ are the regular and coordinate dependent width of the mesa, respectively.

Calculated according to (4) electric field dependencies on y are shown in Figure 3 by a solid line. It is seen, that the used approximation fits well with calculated results. It should be notice that this simple approximation (4) can be used for the mesa in which conducting currents are dominated. When the specific conductivity of the mesa decreases the electric field does not follow the geometry of the mesa.

Considering the other components of the electric field in the mesa it should be pointed out that due to symmetry of the layer E_x component is zero on the

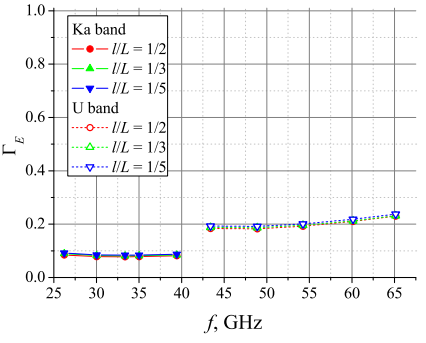


Fig. 4. Dependence of the reflection coefficient from the structure with nicked mesa on frequency within considered frequency range for the different ratio l/L. Calculation parameters: $\varepsilon = 12$, $\sigma = 500$ S/cm, $d = 0.1\mu m$, $f = 33.4$ GHz for Ka-band and $f = 54.3$ GHz for U-band.

y axis. It slightly increases with x whereas the amplitude of the E_z is negligible. In general, the component E_y dominates within the mesa.

Dependencies of the reflection coefficient from the investigated structures on frequency are shown in Figure 4. Calculation results for different shape structures within both frequency bands are presented. It is seen that reflection coefficient in U-band is larger comparing it with the reflection coefficient in Ka-band. It should be noticed that for the structures under investigation the ratio w/a is larger in the U-band and this is a reason for the larger value of the reflection coefficient in this band.

6 Conclusions

The structure consisting of the thin conducting layer with metal contacts inserted in the waveguide has been considered. It has been shown that the strong increase of the electric field appears in the narrowest point of the mesa when the conducting currents dominate. It was found that at this condition the distributions of the electric field depends on the geometry of the mesa and is practically independent of frequency in the considered frequency range. A simple approach, based on the solution of Poisson equation in a DC electric field with the assumption that the drift velocity of the electrons is independent of the electric field, fits well calculated dependences of the electric field in the waveguide for high conductivity mesas.

Acknowledgments. The first author was supported by the Agency for International Science and Technology Development Programmes in Lithuania within the EUREKA Project E!3691 OPTCABLES and by the Lithuanian State Science and Studies Foundation within the project on B-03/2007 "Global optimization of complex systems using high performance computing and GRID technologies".

The last two authors acknowledge the partial their support by the Lithuanian Science and Studies Foundation within the project P-01/2009 "Nanostructures for microwave and terahertz electronics".

References

1. Ašmontas, S.: Electrogradient Phenomena in Semiconductors, Vilnius, Mokslas (1984) (in Russian)
2. Ašmontas, S., Gradauskas, J., Kozic, A., Shtrikmann, H., Sužiedelis, A.: Submicrometric heavily doped n-GaAs structures for microwave detection. Acta Phys. Pol. A 107(1), 147–150 (2005)
3. Ašmontas, S., Gradauskas, J., Sužiedelis, A., Valušis, G.: Submicron semiconductor structures for microwave detection. Microelectronic Engineering 53, 553–556 (2000)
4. Čiegis, R., Čiegis, R., Jakušev, A., Šaltenienė, G.: Parallel variational iterative algorithms for solution of linear systems. Mathematical Modelling and Analysis 12(1), 1–16 (2007)
5. Čiegis, R., Radziunas, M., Lichtner, M.: Numerical algorithms for simulation of multisection lasers by using traveling wave model. Mathematical Modelling and Analysis 13(3), 327–348 (2008)
6. Jakušev, A., Čiegis, R., Laukaitytė, I., Trofimov, V.: Parallelization of linear algebra algoritms using ParSol library of mathematical objects. In: Čiegis, R., Henty, D., Kagstrom, B., Žilinskas, J. (eds.) Parallel Scientific Computing and Optimization. Advances and Applications. Springer Optimization and Its Applications, vol. 27, pp. 25–36 (2009)
7. Kancleris, Ž., Tamošūnas, V., Dagys, M., Simniškis, R., Agee, F.: Interaction of a semiconductor sample partly filling a waveguide's window with milimeter wave radiation. IEE Proc. Microw. Antennas Propag. 152(4), 240–244 (2005)
8. Yee, K.S.: Numerical solution of initial boundary problems involving Maxwell's equation in isotropic media. IEEE Trans. Antennas Propagation 14(3), 302–307 (1966)
9. Laukaitytė, I., Čiegis, R.: Finite-difference scheme for one problem of nonlinear optics. Mathematical Modelling and Analysis 13(2), 211–222 (2008)
10. Maloney, J.G., Smith, G.S.: The efficient modelling of thin material sheets in the FDTD method. IEEE Trans. Antennas and Propagation 40, 323–330 (1992)
11. Seliuta, D., Širmulis, E., Tamošiūnas, V., Balakauskas, S., et al.: Detection of terahertz/ sub-terahertz radiation by asymmetrically-shaped 2DEG layers. Electronics Letters 40(10), 631–632 (2004)
12. Seliuta, D., Kašalynas, I., Tamošiūnas, V., Balakauskas, S., et al.: Silicon lens-coupled bow-tie InGaAs-based broadband terahertz sensor operating at room temperature. Electronics Letters 42(14), 825–827 (2006)
13. Taflove, A., Hagness, S.: Computational Electrodynamics: The Finite-Difference Time-Domain Method. Artech House, Norwood (2000)

Numerical Health Check of Industrial Simulation Codes from HPC Environments to New Hardware Technologies

Christophe Denis

EDF Research and Development,
SINETICS Department,
1 avenue du Général de Gaulle,
92141 Clamart CEDEX, France
Christophe.Denis@edf.fr

Abstract. The numerical health check of industrial codes is crucial to give confidence about the computed results performed by studying the round-off error propagation. This problem is exacerbated in a supercomputing environment where trillions of floating-point operations may be performed every second. A parallel program based on domain decomposition as shown in this paper could compute slightly different results depending on the number of processors. This numerical health check is also needed to verify if a numerical code (or some parts of the numerical code) could still have an acceptable accuracy when using single precision instead of double precision which is useful to run numerical codes on new hardware technologies like GPU where the double precision is unavailable or expensive. The round-off error propagation is measured with the MPFI (interval arithmetic approach) and CADNA (probabilistic approach) libraries.

1 Introduction

The EDF Research and Development direction covers all EDF group businesses. High performance simulation offers a real opportunity for various EDF group businesses that have to deal with increasing complexity and demands. Even if the academic research on HPC deals more often only on performance, the accuracy of sequential and parallel codes developed at EDF R&D is crucial. It is then important to study the effect of the round-off error propagation on the computed results. Several methods exist to perform round-off analysis, for example the inverse analysis method [6]. Among these methods, the CADNA library and MPFI libraries seem to be at our point of view less intrusive in the original code. The CADNA library is an implementation of discrete stochastic arithmetic for code written in FORTRAN, C and C++ [13]. Several libraries implementing the interval arithmetic exist: we use the MPFI library for programs written in C [8]. Preliminary works about the numerical health check by using CADNA for a sequential code has been presented in [11].

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 330–339, 2010.

This paper is organised as follows. The numerical health check tools are presented in Section 2. Section 3 summarises the work done around the numerical validation on new hardware technologies like GPU. On this architecture, the use of the double precision floating point arithmetic can be expensive and the use of single precision speeds up the computing. Nevertheless, it is necessarily to carry out a rigorous numerical verification of the single precision computation to give confidence in the accuracy of the computed results.

Section 4 deals with the communication scheme of the parallel finite element code TELEMAC3D solving the full 3D free surface Navier-Stokes equations. The parallelism is based on domain decomposition and the communication scheme is implemented by using the MPI library. The domain decomposition involves interface nodes duplicated on several processors. Each processor sums on its interface nodes contributions coming from other processors. Unfortunately, round-off errors occurs during the summation and different values could be assigned for the same interface node on different processors. Finally, we present our concluding remarks and our future work.

2 Numerical Health Check Tools

Many simulation programs are large software systems developed to enable virtual experiments to be conducted on some physical system. The development process, extending from the physical world to the mathematical model, then to the computational model and finally to the computer implementation, involves a number of approximations: physical effects may be discarded, continuous functions replaced by discretized ones and real numbers replaced by finite precision representations. In consequence, approximation is woven into the very fabric of scientific software and cannot be eliminated. Unfortunately, despite developments in software engineering, there is every reason to believe that the comment made by Leslie Fox in 1971 [3] is still valid today, *I have little doubt that about 80 per cent of all the results printed from the computer are in error to a much greater extent than the user would believe*. It is incumbent, therefore, on the computational scientist to understand the source and propagation of these errors and to manage them judiciously. The theme of accuracy and reliability in scientific computation has recently been explored and is amplified in [2]. The propagation of these errors must be addressed to avoid the production of computed results with few or no significant digits. In particular, numerical verification is required to give confidence that the computed results are acceptable. Preliminary works about numerical health check of a sequential code has been as done in [11].

Several methods and tools have been developed over the years to analyse round-off error propagation. These include direct analysis, inverse analysis [6], methods based on algorithmic differentiation, interval arithmetic, CESTAC method and randomised interval arithmetic [12]. Among these methods, the CADNA library and MPFI libraries seem to be at our point of view less intrusive in the original code. We present now these two methods.

2.1 The CADNA Library

The CADNA library is an implementation of discrete stochastic arithmetic (DSA), which is based on the CESTAC method, devoted to programs written in ADA, C, C++ and Fortran [13]. Where no overflow occurs, the exact result, r, of any non exact floating-point arithmetic operation is bounded by two consecutive floating-point values R^- and R^+. The basic idea of the method is to perform each arithmetic operation N times, randomly rounding each time, with a probability of 0.5, to R^- or R^+. The computer's deterministic arithmetic, therefore, is replaced by a stochastic arithmetic where each arithmetic operation is performed N times before the next arithmetic operation is executed, thereby propagating the round-off error differently each time. This is the essence of the CESTAC method. The method furnishes us with N samples, R_i, of the computed result R. The value of the computed result, $\overline{R}$, is the mean value of $\{R_i\}$ and the number of exact significant digits in $\overline{R}$, $C_{\overline{R}}$, is estimated using the mean value and the standard deviation of $\{R_i\}$. The computational zero has been defined as follows:

Definition 1. *During the run of a code using the CESTAC method, an intermediate or a final result R is a computational zero, denoted by @.0, if one of the two following conditions holds:*

- $\forall i, R_i = 0$
- $C_{\overline{R}} \leq 0$

Any computed result R is a computational zero if either $R = 0$, R being significant, or R is non significant. In other words, a computational zero is a value that cannot be differentiated from the mathematical zero because of its rounding error.

2.2 The MPFI Library

MPFI is intended to be a portable library written in C for arbitrary precision interval arithmetic with intervals represented using MPFR reliable floating-point numbers [8]. It is based on the GNU MP library and on the MPFR library. The purpose of an arbitrary precision interval arithmetic is on the one hand to get guaranteed results, thanks to interval computation, and on the other hand to obtain accurate results, thanks to multiple precision arithmetic.

The basic principle of interval arithmetic consists in enclosing every number by an interval containing it and being representable by machine numbers: for instance it can be stored as its lower and upper endpoints and these bounds are machine numbers, or as a centre and a radius which are machine numbers. For example, on a radix-10 machine with 3 digits of mantissa, π would be represented by the interval [3.14, 3.15]. The arithmetic operations are extended for interval operands in such a way that the exact result of the operation belongs to the computed interval.

2.3 A First Example: The Precariousness of Relying on Extended Precision

The aim of this example is to present the precariousness of relying on extended precision to give confidence about the computed results. Consider the following innocuous looking function proposed in [9] to be evaluated at $x = 77617$ and $y = 33096$:

$$f(x, y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y) \quad (1)$$

Note that the numerical result could differ if the summation order changed but the purpose of this example is to only underline the round-off errors problem. A Fortran implementation of Eq. (1), executed on an Itanium processor using the Intel ifort compiler, in single, double and quadruple precision produces the results shown in the first three entries of Table 1. The first three entries might lead the unwary to conclude that the single precision result is incorrect and that the double precision result is accurate to 14 decimal digits. Unfortunately, the interval result computed by MPFI (in single or double precision) shows that $f(x, y)$ is very badly computed due to round-off error propagation. As double precision is concerned, $f(x, y)$ is guaranteed to be between $-1.180591621 \times 10^{22}$ and $7.083549725 \times 10^{21}$: the interval is too large. The same phenomenon is observed with CADNA in single or double precision. The result $f(x, y)$ is @.0 that is to say it has no significant digit. A good approximation of the computed results is obtained with MPFI with 130 bits in mantissa ($f(x, y) = -8.2739... \times 10^{-1}$). This example illustrates that round-off error can seriously compromise the reliability of a fixed precision floating-point computation and the importance of round-off error analysis tools to detect this problem.

Table 1. The computation of $f(x, y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + x/(2y)$

Method	$f(77617, 33096)$
Fortran:single precision	6.3382530×10^{29}
Fortran:double precision	1.17260394005318
Fortran:quad precision	1.17260394005317863185883490452018
MPFI:single precision	$[-8.239728902 \times 10^{30}$, $8.239729506e30]$
MPFI: double precision	$[-1.180591621e22$, $7.083549725e21]$
CADNA:single precision	@.0
CADNA:double precision	@.0
MPFI:"nearby exact" value (130 bits in mantissa)	$[-8.273960600 \times 10^{-1}$, $-8.273960599 \times 10^{-1}]$

3 Impact on the Accuracy by Using Single Precision Instead of Double Precision

The new multi-core architectures with reduced clock frequencies like GPU provide an high power computing power for a relatively low power consumption. In these architectures, the double precision could be unavailable or expensive. On CPU, the use of single precision reduces the computing time and the amount of memory required. It is then important to estimate the number of significant digits obtained with single precision. We have worked in this topic during a research fellow position on a financial program written in C based on Monte-Carlo computations. It returns the value v of the portfolio of swaptions. Only four significant digits are required for v. More information about the LIBOR code could be found in [14]. The CADNA and the MPFI libraries has been implemented in the LIBOR code. The code has been performed by using single and double precision. Table 2 reports the values of v by using the CADNA and MPFI libraries with the single and double precision. Only the significant digits given by CADNA are printed in these two tables.

Table 2. The values of v by using the CADNA and MPFI libraries in single and double precision

Initial Precision	Number of samples	v (with CADNA)	v (with MPFI)
Double precision	1000	2.2432331204357e2	[2.2432331204356563e2, 2.2432331204357712e2]
Double precision	10000	2.2432331204357e2	[2.2432331204356557e2, 2.2432331204357718e2]
Single precision	1000	2.243232e2	[2.24320236e2, 2.24326386e2]
Single precision	10000	2.243232e2	[2.24320236e2, 2.24326371e2]

For all results presented in Table 2, the results given by CADNA are inside the interval computed by MPFI. The number of samples does not influence the accuracy of the computed results. The number of significant digits given by CADNA of the results computed with double precision is 14. This number decreases to 6 by using single precision. Using single precision is then sufficient as four significant digits are required for v.

4 Domain Decomposition and Round-Off Errors

The domain decomposition is widely used to solve in parallel numerical applications. The domain decomposition is a divide and conquer strategy:

1. The domain (finite element mesh or finite volume mesh) is divided into N_s subdomains.
2. Each subdomain is assigned to a processor and local computations are performed on it.

3. Communications are performed to gather local computations located at the interface between subdomains.

The communication algorithm has been here explicitly developed but for example the Parsol library could be used to build parallel linear algebra algorithms [7]. The numerical results have to be identical on interface nodes among subdomains. It is not always the case as round-off errors occur in the context of floating point computations. Consequently, a parallel code based on a domain decomposition could give some slightly different results in contrast with the sequential code. The objective of this section is to enlight this phenomena by taking as example the parallel finite element code TELEMAC3D. It is a joint work done with E. Razafindrakoto from the EDF National Hydraulics and Environment Laboratory. The section is organised as follows. Firstly, the TELEMAC3D code is briefly described. After describing its communication scheme, we show why it could produce different values on interface nodes among subdomains. These differences could produce deadlocks between processors in particular during the computing of derivatives on interfaces nodes. A development modifying the communication scheme has been made to avoid these numerical differences. Unfortunately, even if differences disappear, this development does not ensure the accuracy of the results. The last point of this section presents the round-off error propagation effects on the original and modified communication scheme.

4.1 The TELEMAC3D Code

TELEMAC3D is a program contained in the TELEMAC system designed and used by the EDF National Hydraulics and Environment Laboratory. The aim of TELEMAC system is to study the numerical modelling system for free surface hydrodynamics, sedimentology, water quality waves and underground flows. TELEMAC3D solves the full 3D free surface Navier-Stokes equations. It is mostly based on the finite element method and the basic principles of the solution procedure are detailed in [4]. The finite element mesh is composed of super-imposition of 2D triangles meshes so that the 3D finite element is a prism. The upper level follows the free surface. The parallelism is based on domain decomposition and the communication scheme is implemented by using the MPI library. The parallelism in TELEMAC3D is based on the message passing paradigm. The finite element mesh is divided into N_s subdomains $SD^{(j)}$ without finite element overlapping by using the graph partitioning tool METIS. Due to the domain decomposition, there exist nodes - called interface nodes - shared by several subdomains. Alg. 1 summarises the two main steps of the TELEMAC3D parallel version: the computation and the communication steps.

4.2 Problem of the Communication Scheme and First Solution

At the end of each communication step, entries of V must have the same values among subdomains.

Consider a node a shared by four subdomains. Each subdomain $SD^{(j)}$ has a one entry V_a^j corresponding to the node a. During the communication step,

for all timestep t **do**
{**Computation step**}

New values of the nodes - including interface nodes - are locally computed in parallel on each subdomain $SD^{(j)}$. Depending of the type of computation, one to three contributions are computed and stored into one to three arrays V_1, V_2, V_3. Without loss of generality, we consider in the rest of the paper that there is only one contribution per node stored in V. Entries of V correspond to internal nodes or interface nodes.

{**Communication step**}

The entries of V corresponding to interface nodes scattered on several subdomains need to be gathered. The gather operation could be the sum, the maximum value, the minimum value or the maximum absolute value.

end for

Algorithm 1. Main steps of the TELEMAC3D parallel version

each subdomain receive the value of this node coming from the three other subdomains. These values will be successively added with the local value. Each subdomains could compute in parallel these sums in different order depending on the communication network :

$$V_a^1 \leftarrow V_a^1 + V_a^2 + V_a^3 + V_a^4,\ V_a^2 \leftarrow V_a^2 + V_a^1 + V_a^3 + V_a^4,$$
$$V_a^3 \leftarrow V_a^3 + V_a^1 + V_a^2 + V_a^4,\ V_a^4 \leftarrow V_a^4 + V_a^1 + V_a^2 + V_a^3$$

Unfortunately, as floating point computation is concerned, the result of the sums could not be the same on each subdomain due to round-off errors. The floating point sum is not associative. The first solution given was to assign on each interface node the maximum value of the sums among subdomains. The drawbacks of this solution are the increase of the communication volume and the problem of round-off errors is hidden but still exists. Indeed, the maximum value of the sum is not necessarily the solution with few round-off errors. The communication phase has been modified in order to compute the sum in the same order among subdomains (the ascending order of MPI process). This modification allows to avoid computing the maximum value. The gain obtained on the computing time is from 2.5% (1 024 subdomains) to 9% (2 048 subdomains).

5 Measuring the Accuracy of the Floating Point Summation

The CADNA library has been implemented in the TELEMAC3D communication scheme to control the accuracy of the floating point summation. The test case is a numerical study having a finite element mesh with 6 352 954 3D nodes and 12 083 160 3D finite elements. The number of time steps of TELEMAC3D has been

Table 3. Number of significant digits obtained during the summation by using 1024 to 8192 MPI process on BG/P

1 024 subdomains		
nsd	Original communication scheme	Ascending order of MPI processes
	Percentage of sums having *nsd* significant digits	Percentage of summation having *nsd* significant digits
15	96.5%	97.1%
14	2.98%	2.68%
13	0.27%	0.23%
12	0.3%	0.03%
11	0.01%	0.003%
10	0.0002%	0.0002%
9	0.0002%	0%
2 048 subdomains		
nsd	Original order	Ascending order of MPI processes
	Percentage of sums having *nsd* significant digits	Percentage of summation having *nsd* significant digits
15	96.9%	96.7%
14	2.85%	2.96%
13	0.27%	0.3%
12	0.03%	0.03%
11	0.003%	0.004%
10	0.0006	0.0002%
9	0	0
4 096 subdomains		
nsd	Original order	Ascending order of MPI processes
	Percentage of sums having *nsd* significant digits	Percentage of summation having *nsd* significant digits
15	96.1%	95.6%
14	2.76%	3.8%
13	0.91%	0.3%
12	0.22%	.04%
11	0.02%	0.002%
10	0.00045%	0.0002
9	0	0
8 192 subdomains		
nsd	Original order	Ascending order of MPI processes
	Percentage of sums having *nsd* significant digits	Percentage of summation having *nsd* significant digits
15	97,6%	97,39%
14	2.23%	2.23%
13	0.19%	0.21%
12	0.02%	0.02%
11	0.002%	0.002%
10	0.0002%	0.0002%
1	0.0005%	0

fixed to 1000. The test case has been run from 1 024 to 8 192 processors on the EDF IBM BG/P supercomputer. Table 3 represents the percentage of summation having nsd significant digits for the original and modified communication scheme. The total number of summation during a computing is about 400000. Not surprisingly, the summation performed in original order and the summation made in the ascending order of MPI processes produce both round-off errors. Moreover, the number of instabilities does not increase with the number of subdomains. In a large number of cases (more than 96% of the number of sums), there is no round-off error: the sum having the same magnitude. The number of significant digits decreased - for one case the sum has only one significant digit - when the magnitude of operands differs. Even if the modified communication scheme permits to have the same value on interface nodes among subdomains, round-off error problems remains. The future work consist to implement and evaluate with CADNA error-free transformation for the sum of two floating point numbers [10].

6 Conclusion and Future Work

The results presented in the paper underline that numerical health tools like CADNA is useful to give confidence about the computed results. It is more important in the context of industrial codes than the speed of the computing.

A numerical health check permits also to know if the single precision could be used instead of the double precision. This is important to reduce the amount of memory required by a program: why using double precision if the single precision give satisfactory results ? This is also more important on new hardware technologies like GPU where the double precision is unavailable or expensive. In the context of parallel numerical code, these experimental results show that domain decomposition could produce round-off errors even if the sequential program is accurate. The future work is to implement error-free transformation for the floating sum in the TELEMAC3D communication scheme. This work will be also tested on other EDF parallel codes based on domain decomposition.

References

1. Alt, R., Lamotte, J.-L.: Experiments on the evaluation of functional ranges using random interval arithmetic. Mathematics and Computers in Simulation 56(1), 17–34 (2001)
2. Einarsson, B. (ed.): Accuracy and Reliability in Scientific Computing. SIAM, Philadelphia (2005)
3. Fox, L.: How to get meaningless answers in scientific computation (and what do about it), IMA Bulletin (1971)
4. Hervouet, J.-M.: Hydrodynamics of Free Surface Flows: Modelling with the Finite Element Method, Xiley (April 2007), IBSN : 978-0-470-03558-0
5. Higham, N.J.: The accuracy of floating point summation. SIAM Journal on Scientific Computing (1993)

6. Higham, N.J.: Accuracy and Stability of Numerical Algorithms. SIAM, Philadelphia (1996)
7. Jakušev, A., Čiegis, R., Laukaitytė, I., Trofimov, V.: Parallelization of linear algebra algorithms using ParSol library of mathematical objects. In: Čiegis, R., Henty, D., Kagstrom, B., Žilinskas, J. (eds.) Parallel Scientific Computing and Optimization. Advances and Applications. Springer Optimization and Its Applications, vol. 27, pp. 25–36 (2009)
8. Revol, N., Rouillier, F.: Motivations for an arbitrary precision interval arithmetic and the mpfi library. Reliable Computing 11(4), 275–290 (2005)
9. Rump, S.M.: Reliability in Computing. In: The Role of Interval Methods in Scientific Computing. Academic Press, London (1998)
10. Rump, S.M., Ogita, T., Oishi, S.: Accurate Floating-point Summation Part I: Faithful Rounding. SIAM Journal on Scientific Computing (SISC) 31(1), 189–224 (2008)
11. Scott, N.S., Jezequel, F., Denis, C., Chesneaux, J.-M.: Numerical 'health check' for scientific codes: the CADNA approach. Computer Physics Communications 176(8), 507–521 (2007); Scientific Computing, Academic Press (1998)
12. Zilinskas, J., Bogle, I.D.L.: Evaluation ranges of functions using balanced random interval arithmetic. Informatica 14(3), 403–416 (2003)
13. CADNA: Control of Accuracy and Debugging for Numerical Applications. Université Pierre et Marie Curie, Paris, http://www.lip6.fr/cadna
14. http://people.maths.ox.ac.uk/~gilesm/hpc/
15. The TELEMAC system, http://www.telemacsystem.com

Application of Parallel Technologies to Modeling Lithosphere Dynamics and Seismicity*

Boris Digas, Lidiya Melnikova, and Valerii Rozenberg

Institute of Mathematics and Mechanics,
Ural Branch of Russian Academy of Sciences,
S. Kovalevskoi Str. 16, 620219 Ekaterinburg, Russia
{digas,meln,rozen}@imm.uran.ru
http://www.imm.uran.ru

Abstract. A problem of modeling some processes in the crust is considered. A brief description of different modifications of the spherical block model of lithosphere dynamics and seismicity is presented; the emphasis is on incorporation of lithospheric inhomogeneities into the model. An approach to program realization based on the use of parallel technologies is applied. Results of numerical experiments with an approximation of the global system of tectonic plates are discussed.

Keywords: Block models, tectonic plates, earthquake catalogs.

1 Introduction

Study of earthquakes with the statistical and phenomenological analysis of real catalogs has the disadvantage that the reliable data cover, in general, a time interval of about one hundred years or less, which is very short in comparison with the duration of tectonic processes responsible for the seismic activity. Therefore, the patterns of the earthquake occurrence identifiable in a real catalog may be only apparent and may not repeat in the future. In this connection, mathematical models of seismicity, i.e., of temporal-spatial earthquake sequences, are important tools that yield synthetic catalogs, which may cover a very long time interval in order to acquire, by means of appropriate optimization procedures, more reliable estimates of parameters of a seismic flow and to search for premonitory patterns preceding large earthquakes [2]. Every event in a synthetic catalog is characterized by a time moment, epicenter coordinates, a depth, a magnitude, and, for some models, an intensity. Simulation of lithosphere dynamics allows us to obtain velocity fields for different depths, forces, displacements induced by these forces, as well as characteristics of the structural interaction.

* The work was performed within the framework of the Program of Presidium of Russian Academy of Sciences "Intellectual information technologies, mathematical modeling, system analysis, and automatization" and was supported by the Russian Foundation for Basic Research (Project 09-01-00378) and by the Ural-Siberian Interdisciplinary Project.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 340–349, 2010.

There exist many different approaches to modeling lithospheric processes (see, e.g., [2] and its bibliography); nevertheless, we can mark out the two main directions. The first direction is based on detailed investigation of one specific tectonic fault, or, rather often, one strong earthquake in order to reproduce certain pre- and/or post-seismic phenomena (relevant to this fault or event). In contrast, models of the second direction developed relatively recently treat the seismotectonic process in rather abstract way; the main goal of simulation is reproducing general universal properties of observed seismicity (primarily, the Gutenberg – Richter law on frequency-magnitude relation, clustering, migration of events, seismic cycle and so on). However, it seems that an adequate model should reflect some universal features of nonlinear systems as well as the specific geometry of interacting faults. The block models of lithosphere dynamics and seismicity (see, for example, [9,10]) have been developed with both requirements taken into account. This approach to modeling is based on the representation of tectonic plates as a system of perfectly rigid blocks, which is in a quasi-static equilibrium state; model events are stress-drops occurring on faults separating blocks under the action of external forces. The plane block model [9, 10], in which a block structure is bounded by two horizontal planes, has been the most extensively studied. Approximations of real seismic regions have been built on its basis. But while trying to simulate the motion of a system of global tectonic plates with the plane block model, significant distortions were revealed. In order to overcome them, the spherical geometry has been involved [8]. The present paper, being a continuation of the investigations [6,8], actually represents the brief review of designed modifications of the spherical block model and the discussion on some results of numerical experiments. One of the goals of this work is to demonstrate that there is a natural succession in algorithmic and program realization, including parallelization, of plane and spherical block models, despite the fact that the latter is much more complicated.

2 Different Modifications of the Spherical Block Model

Let us briefly describe the spherical block model of lithosphere dynamics and seismicity.

A block structure is a limited and simply connected part of the spherical layer of depth H bounded by two concentric spheres. The outer sphere represents the Earth's surface and the inner one represents the boundary between the lithosphere and the mantle. The partition of the structure into blocks is defined by faults intersecting the layer. Each fault is a part of a cone surface inclined at a certain angle to the outer sphere. Faults intersect along curves, which meet the outer and inner spheres at points called vertices. Fragments of faults limited by adjacent vertices are called segments. The common parts of blocks with the limiting spheres are spherical polygons, those on the inner sphere are called bottoms. A block structure may be a part of the spherical shell and be bordered by boundary blocks, which are adjacent to boundary segments. Another possibility is to consider the structure including the whole spherical shell (covering the

whole surface of the Earth) without boundary blocks. All block displacements are considered as negligible, compared with block sizes. Therefore, the geometry of the block structure does not change during the simulation, and the structure does not move as a whole. All blocks have six degrees of freedom. The displacement of each block consists of translation and rotation components. The motions of the boundary blocks and of the underlying medium are assumed to be known; they are described as rotations on the sphere, i.e., axes of rotation and angular velocities are given.

Depending on the way of treating the depth of the spherical layer, several modifications of the model are worked out. In the first modification (without depth) [8], it is assumed that all characteristics of points belonging to the block structure are determined only by their coordinates and do not depend on the depth, since this depth is significantly less than the linear dimensions of blocks. The modification with constant depth [6] uses the idealized assumption that the lithosphere is homogeneous (in the sense that all the blocks have the same depth H, and all parts of a block/a fault have the same properties). The modification with varying depth announced in [6] and being under development now provides a possibility of specifying different depths for different blocks and of changing fault parameters depending on its depth. Note that this is the first attempt of taking into account the lithospheric inhomogeneities (e.g., differences in the structure of continental and oceanic crust) within the framework of block models.

Because the blocks are perfectly rigid, all deformations take place in the faults and at the block bottoms; forces arise on the inner sphere due to displacements of blocks with respect to the underlying medium and on fault surfaces due to displacements of neighboring blocks. The elastic force per unit area (f_t, f_l, f_n) is defined by

$$f_t = K_t(\Delta_t - \delta_t), \quad f_l = K_l(\Delta_l - \delta_l), \quad f_n = K_n(\Delta_n - \delta_n). \tag{1}$$

Here (t, l, n) is the coordinate system connected with the point of application of the force (the axes t and l lie on the plane tangent to the fault's surface, the axis n is perpendicular to it); Δ_t, Δ_l, Δ_n are components of the relative displacement in the system (t, l, n) of the neighboring blocks or of the block and the underlying medium of the neighbor; δ_t, δ_l, δ_n are corresponding inelastic displacements, the evolution of which is described by the equations

$$\frac{d\delta_t}{dt} = W_t f_t, \quad \frac{d\delta_l}{dt} = W_l f_l, \quad \frac{d\delta_n}{dt} = W_n f_n. \tag{2}$$

The coefficients K_t, K_l, and K_n (1), characterizing the elastic properties of the fault, and the coefficients W_t, W_l, and W_n (2), characterizing the viscous properties of the fault, may be different for different faults and, in addition, may depend on the depth.

The formulas for calculating forces and inelastic displacements on block bottoms are obtained by analogy. Components of the translation vectors of any inner block and angles of its rotations are found from the condition that the total force and the total moment of forces acting on the block are equal to zero.

This is the condition of quasi-static equilibrium of the system. Since in the model we consider the linear dependence of forces and moments on displacements and rotations of blocks, the system of equations for determining these values must be also linear:

$$\mathbf{A}\mathbf{w} = \mathbf{b}. \tag{3}$$

Here, the components of the unknown vector $\mathbf{w} = (w_1, w_2, ..., w_{6n})$ are the components of translation vectors of inner blocks and angles of their rotation (n is the number of such blocks). The matrix $\mathbf{A}$ (of dimension $6n \times 6n$) does not depend on time and can be calculated only once, at the beginning of the process. To calculate various curvilinear integrals, one should discretize (divide into cells) the spherical surfaces of the block bottoms and fault segments. The values of forces and inelastic displacements are assumed to be equal for all points of the cell. System (3) is solved at discrete time moments t_i.

Calculating the force acting on a fault, we find the ratio of the stress to the pressure by the formula

$$\kappa = \frac{\sqrt{f_t^2 + f_l^2}}{P - f_n}. \tag{4}$$

Here P is the parameter, which may be interpreted as the difference between the lithostatic and the hydrostatic pressure. The interaction between the blocks (between the block and the neighboring underlying medium) is visco-elastic (a "normal state"), so long as the value κ (4) at the fault separating the elements of the structure is below a certain strength level. If, at some moment, a critical value is reached (the admissible thresholds are a priori specified and may be different for different faults) then, in accordance with the dry friction model, a stress-drop (a "failure") occurs through abrupt changing the inelastic displacements δ_t, δ_l, and δ_n. The failures represent earthquakes. Immediately following the earthquake for some period of time (so called healing time), the corresponding parts of the faults are in a "creep state". This state differs from the normal one because of the more rapid growth of inelastic displacements and continues until the stress falls below a given level.

A synthetic earthquake catalog is produced as a main result of the simulation process. All cells of the same fault, in which the failures occurred at the same time t_i, are considered as a single earthquake. The parameters of the earthquake are defined as follows: (a) the time of the event is t_i; (b) the epicentral coordinates and the depth are the weighted sums of the corresponding coordinates and the depths of the cells involved in the earthquake (the weight of each cell is given by its area divided by the sum of areas of all cells involved in the earthquake); (c) the magnitude is calculated by the known in seismology formula [11]:

$$M = 0.98 \lg S + 4.07, \tag{5}$$

where S is the total area of cells (in km^2) involved in the earthquake. In addition, the model allows us to obtain the instantaneous kinematics of blocks and the information on their interaction along the boundaries. It should be noted that formulas (1)–(5) look like corresponding formulas for the plane model (compare

with [10]) but all the forces and displacements are to be calculated on the sphere. For a detailed description of the block models, see, e.g., [6,9].

3 Parallelization: Efficiency and Scalability Analysis

The computational experiments show that the spherical block model of lithosphere dynamics and seismicity during performing on sequential computers requires very considerable expenditures of memory and processor time. Due to this reason, the problem of simulating dynamics for structures with a large number of blocks and small enough steps of the temporal-spatial discretization needs special solving tools. However, the approach applied to modeling admits effective parallelization of calculations on a multiprocessor machine. The standard scheme "master-worker" ("processor farm") successfully used in the plane model [10] is applied to the spherical model as well. The flowchart of the main calculative procedure looks exactly the same as in [10]. Let us give some explanations. In the beginning of the work, a specific processor, which the program has loaded to, is detected by its number (zero number becomes the master). Then the information on the block structure is read, and auxiliary calculations (the space discretization, calculation of the matrix $\mathbf{A}$ of system (3)) are performed. At every time step, the most time-consumable procedure is calculation of the values of forces and inelastic displacements in all cells of the block bottoms and fault segments. Since these calculations can be performed independently from each other, it is necessary to share them uniformly between all processors. Note that, calculating all the values above at its own portion of cells, the master plays the role of the worker as well. The key elements of the parallel algorithm in question are the uniform distribution of cells and the optimal exchange between processors. The information exchange is realized according to the following scheme. At every time step, the master calculates the new values of block, boundary block, and underlying medium displacements (it requires insignificant time due to the small dimension of system (3)), then necessary parameters are transferred to the workers. After recalculation of the forces and inelastic displacements, the workers return their parts of the vector $\mathbf{b}$ to the master, then the next time step is carried out. For processing the situation treated as an earthquake, the scheme is slightly complicated, since in this case the master should ask all the workers until cells of segments in the critical state exist; then the master receives the information on model events and writes it into a file of special structure. For such a scheme, the time of calculations on each processor is much more than the time of exchange. Therefore, rather high useful loading of each processor is achieved.

The simulation was performed at the Joint Supercomputer Center (Moscow, Russia; Supercomputer MVS-15000M, 574 processors 2 $\times$ PowerPC 970 (2.2 GHz), the peak performance is about 10 TFlops) and at the Institute of Mathematics and Mechanics (Ekaterinburg, Russia; UM64 machine, 108 processors Opteron (2.6 GHz), the peak performance is about 560 GFlops). In addition, on UM64 machine, the experiments for testing the dependence of time necessary

for solving the problem on the number of processors were carried out. A time-taking variant with a considerable number of earthquakes occurred was chosen; 100 time steps were considered (note that a typical variant needs 20 000 steps). The speedup, the granularity level, and the efficiency were analyzed. The results of testing are presented in Table 1.

It follows from Table 1 that for $p \leq 20$ the speedup S_p is slightly less than p, the efficiencies E_p^1 and E_p^2 are rather high, not less than 0.9. The value E_p^2, which is calculated without executing the sequential algorithm, appropriately approximates E_p^1; this is in agreement with theoretical results [5]. It is important that, after introducing the spherical geometry, the volume of calculations essentially increases, whereas the exchanges between processors remain almost the same. Therefore, the share of calculation time (in the total time) increases with respect to the share of idle and exchange time; this results in the efficiency increase comparing with the plane model [10].

Table 1. Dependence of speedup and efficiency on the number of processors

p	t_{calc}	t_{exch}	t_{tot}	S_p	E_p^1	G	E_p^2
1	6335.84	–	6335.84	–	–	–	–
2	3256.52	34.60	3291.12	1.92	0.96	94.12	0.99
4	1626.94	38.52	1665.46	3.80	0.95	42.24	0.98
8	814.74	33.98	848.72	7.46	0.93	23.98	0.96
10	654.34	27.93	682.27	9.29	0.93	23.43	0.96
16	417.36	22.26	439.62	14.41	0.90	18.75	0.95
20	331.23	22.33	353.56	17.92	0.90	14.83	0.94
28	237.08	28.53	264.61	23.94	0.86	8.31	0.89

Notation: p is the number of processors, t_{calc} is calculation time, t_{exch} is idle and exchange time, t_{tot} is total expenditure time (all in seconds), $S_p = T_1/T_p$ is speedup, T_1 is performance time for sequential algorithm, T_p is performance time for parallel algorithm on p processors, $G = t_{\text{calc}}/t_{\text{exch}}$ is granularity level, $E_p^1 = S_p/p$, $E_p^2 = G/(G+1) = t_{\text{calc}}/t_{\text{tot}}$ are efficiencies (the last one is also called processor utilization).

All the times presented above are measured on the master and, therefore, correspond to the most time-taking process. Note that since the master also plays the role of the worker, its idle time can be considered as negligible. Some fluctuations of the exchange time (when the number of processors is increased) are possibly explained by the architecture of the machine and/or by the specificity of send/receive procedures in use. Due to the fact that all the processors are almost uniformly loaded with respect to calculations, it seems that the approach applied here to estimate the efficiency is valid.

Now pass to some theoretical analysis of the parallel algorithm. We are interested in evaluating the scalability of the algorithm in the following sense: how the problem size (the amount of necessary calculations) must scale with the number of processors to keep the efficiency constant [1]. Toward this aim, we consider an idealized performance model introducing different characteristics of the total expenditure time $t_{\text{tot}} = t_{\text{calc}} + t_{\text{exch}}$.

Let us estimate the calculation time in a single time step as the time necessary for calculations at spatial cells (the rest job, including solving system (3), requires quite negligible resources):

$$t_{\text{calc}} = t_{\text{sg}} N_{\text{sg}} + t_{\text{bl}} N_{\text{bl}},$$

where t_{sg} and t_{bl} are the average calculation times at a single cell of segment and block bottom discretization; N_{sg} and N_{bl} are the numbers of segment and block bottom cells, respectively. The last two parameters determine the problem size for a fixed block structure. To estimate the exchange time in a single time step, we introduce the time t_{msg} required to send (to receive) a message of size L words:

$$t_{\text{msg}} = t_{\text{s}} + t_{\text{w}} L,$$

where t_{s} is the message startup time, t_{w} is the transfer time per word (both are machine parameters). In the algorithm, $L = An_{\text{bl}} + B$ (here n_{bl} is the number of blocks in the structure, A and B are constants, which can be explicitly written). Hence, the exchange time for p processors (the master and $p-1$ workers), with taking into account send/receive procedures, is

$$t_{\text{exch}} = 2(p-1)(t_{\text{s}} + t_{\text{w}}(An_{\text{bl}} + B)).$$

The performance times T_1 and T_p can be calculated as follows:

$$T_1 = t_{\text{calc}} = t_{\text{sg}} N_{\text{sg}} + t_{\text{bl}} N_{\text{bl}},$$

$$T_p = t_{\text{calc}}/p + t_{\text{exch}} = (t_{\text{sg}} N_{\text{sg}} + t_{\text{bl}} N_{\text{bl}})/p + 2(p-1)(t_{\text{s}} + t_{\text{w}}(An_{\text{bl}} + B)).$$

From the above it follows that the efficiency is

$$E_p^1 = T_1/pT_p = \frac{t_{\text{sg}} N_{\text{sg}} + t_{\text{bl}} N_{\text{bl}}}{t_{\text{sg}} N_{\text{sg}} + t_{\text{bl}} N_{\text{bl}} + 2p(p-1)(t_{\text{s}} + t_{\text{w}}(An_{\text{bl}} + B))}.$$

Thus, the efficiency (a) decreases with increasing p, t_{s}, t_{w}, and n_{bl}; (b) increases with increasing t_{sg}, t_{bl}, N_{sg}, and N_{bl}.

For the constant efficiency, we need to obtain $T_1 \approx E_p^1 pT_p$, i.e.,

$$t_{\text{sg}} N_{\text{sg}} + t_{\text{bl}} N_{\text{bl}} \approx E_p^1 (t_{\text{sg}} N_{\text{sg}} + t_{\text{bl}} N_{\text{bl}} + 2p(p-1)(t_{\text{s}} + t_{\text{w}}(An_{\text{bl}} + B))).$$

In order to make this relation p-insensitive, it is necessary to choose the dependencies $N_{\text{sg}} = c_1 p(p-1)$ and $N_{\text{bl}} = c_2 p(p-1)$, where c_1 and c_2 are some constants. In this case, we obtain the condition under which the efficiency remains constant with increasing p:

$$c_1 t_{\text{sg}} + c_2 t_{\text{bl}} \approx E_p^1 (c_1 t_{\text{sg}} + c_2 t_{\text{bl}} + 2t_{\text{s}} + 2t_{\text{w}}(An_{\text{bl}} + B)).$$

So, for a fixed block structure, increasing the numbers of segment and block bottom cells (thereby improving the accuracy of calculations) proportionally to $p(p-1)$, we obtain the constant efficiency independent from p.

4 Some Results of Numerical Simulation

In this section, some results of application of the spherical block model to studying dynamics and seismicity of the global system of tectonic plates covering the whole surface of the Earth are presented. The structure contains 15 plates (see Fig. 1), for which the following notation is used: NA – North America, SA – South America, N – Nazca, Af – Africa, Ca – Caribbean, Co – Cocos, P – Pacific, S – Somalia, Ar – Arabia, E – Eurasia, I – India, An – Antarctica, Au – Australia, Ph – Philippines, F – Juan de Fuca. The motion of the underlying medium (this is the only reason of the motion of the closed block structure in the case when boundary blocks are absent) is specified as the rotation on the sphere according the model of plate kinematics HS3-NUVEL-1 [4]. The model plate depths are chosen with allowance for distribution of real seismicity in depth.

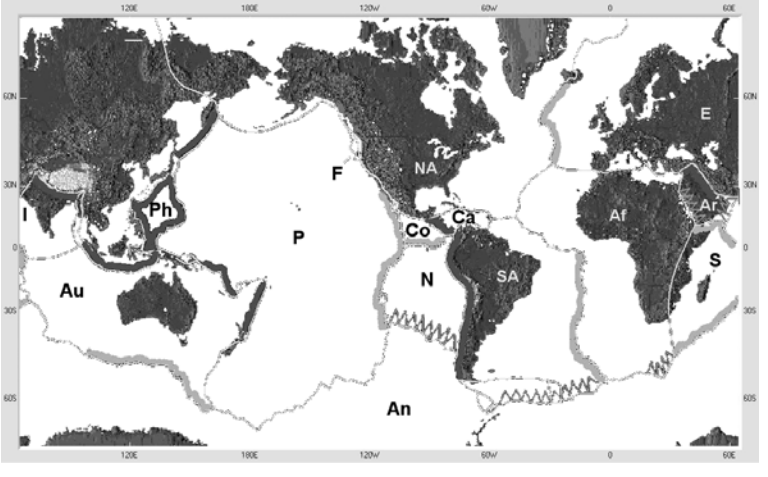


Fig. 1. The global system of tectonic plates and results of simulation of the character of plate boundaries: divergent plate boundaries (spreading, light shading), convergent plate boundaries (subduction, dark shading), transform plate boundaries (sliding, toothed shading)

The behavior of boundary points belonging to plate boundaries, for which one of three types (divergent, convergent, and transform) is clearly marked, is investigated. Such characteristic boundaries [7] (e.g., as South America/Nazca, Pacific/Nazca, South America/Africa, India/Eurasia, surrounding Philippines, etc.) are considered. By means of two displacements of a boundary point in the coordinate system connected with this point as a point of right and left blocks, respectively, its relative displacement was computed. Relative displacements of boundary points characterize qualitatively the interaction between plates along their boundaries, thereby allow us to mark the boundary types (see Fig. 1),

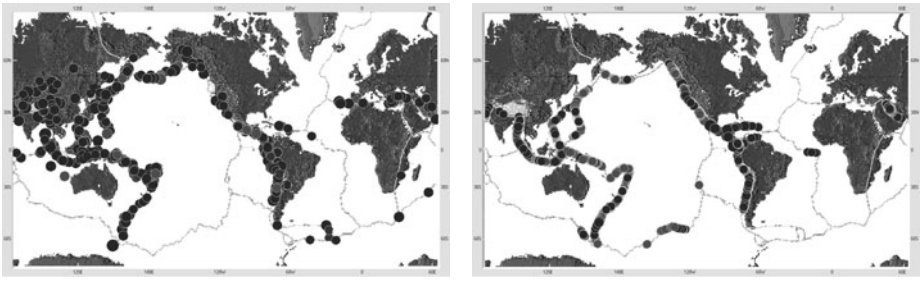


Fig. 2. Epicenters of strongest earthquakes with $M \geq 7.5$; NEIC (left) and synthetic (right) catalogs

which are rather similar to real ones [7]. As a main parameter characterizing the quality of simulation, the spatial distribution of the strongest earthquakes is considered. The comparative analysis of the synthetic catalogs that are composed by means of formula (5) and the real one extracted from the global catalog NEIC [3] and containing events with the magnitude $M \geq 7.5$ for the time period 01.01.1900–31.12.2008 without any restrictions by depth and area of location (Fig. 2) is performed. Note that, since model events occur only on the block boundaries, we cannot obtain earthquakes inside tectonic plates. For all the modifications of the spherical block model, the synthetic catalogs (see one of them (the case of varying depth) in Fig. 2) reflect the most important patterns of global seismicity, namely: (a) two large seismic belts, the circum-Pacific and Alpine-Himalayan (the first is more well-defined), where most of the strong earthquakes occur; (b) extensive, but less pronounced seismicity at mid-oceanic ridges; (c) increased seismic activity associated with triple junctions of plate boundaries. As the most active seismic regions, one can point out such boundaries as Nazca/South America, Cocos/Caribbean, India/Eurasia, California region, Arabia/Eurasia, south-east, east, north-east and, especially, north of Australia, and the Philippine plate margin. The level of synthetic seismicity is extremely small at such boundaries as south of Pacific plate, Nazca/Pacific, east and south-west of Africa, India/Australia, North America/Eurasia. These locations agree in principle with observations (compare the diagrams in Fig. 2); this fact indicates a rather high degree of adequacy of the model.

At the same time, in some regions with rather high seismicity we did not obtain strong earthquakes in all the variants (e.g., at the boundary Africa/Eurasia). Our conjecture is that the motion of global tectonic plates is not a main driving force of seismic activity in these regions. In addition, we analyzed other characteristics, including parameters of the Gutenberg – Richter law on frequency-magnitude relation and the distribution of earthquakes with respect to depth. The modification with varying depth produced a better (in all the aspects listed above) synthetic catalog. This happened as a consequence of the model improvement, without any parameter fitting.

5 Conclusions

Simplifications accepted in the model give no opportunity to draw conclusions on the correspondence between observed and synthetic seismicity at any specific point or in relatively small regions. However, some similarity of the model results and the real data in the global scale is certainly a positive fact; it stimulates a further development of the model. The most perspective is the modification with varying depth, being an attempt of taking into account the lithospheric inhomogeneities within the framework of the spherical block model.

In the paper, we demonstrated that there was a natural succession in algorithmic and program realization of the plane and spherical block models. Concerning parallelization, it is important that, after introducing the spherical geometry, the volume of calculations essentially increased, whereas the exchanges between processors remained almost the same; this resulted in the efficiency increase comparing with the plane model.

Note that the parallel algorithm used in the problem provides an opportunity to consider it as a test problem for approbating new distributed computing environments.

References

1. Foster, I.: Designing and Building Parallel Programs (1995), http://www.mcs.anl.gov/~itf/dbpp/
2. Gabrielov, A.M., Newman, W.I.: Seismicity Modeling and Earthquake Prediction: a Review. Geophysical Monograph 83 18, 7–13 (1994)
3. Global Hypocenters Data Base CD-ROM. NEIC/USGS, Denver, CO (2008)
4. Gripp, A.E., Gordon, R.G.: Young Tracks of Hotspots and Current Plate Velocities. Geophys. J. Int. 150, 321–361 (2002)
5. Kwiatkowski, J.: Evaluation of Parallel Programs by Measurement of its Granularity. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2001. LNCS, vol. 2328, pp. 145–153. Springer, Heidelberg (2002)
6. Melnikova, L.A., Rozenberg, V.L.: Spherical Block Model of Lithosphere Dynamics and Seismicity: Different Modifications and Numerical Experiments. Proceedings of IMM UB RAS 13(3), 95–120 (2007) (in Russian)
7. Mutter, J.C.: Seismic Images of Plate Boundaries. Sci. Amer. 254, 66–75 (1986)
8. Rozenberg, V.L., Sobolev, P.O., Soloviev, A.A., Melnikova, L.A.: The Spherical Block Model: Dynamics of the Global System of Tectonic Plates and Seismicity. Pure Appl. Geophys. 162, 145–164 (2005)
9. Soloviev, A.A., Ismail-Zadeh, A.T.: Models of Dynamics of Block-and-Fault Systems. In: Keilis-Borok, V.I., Soloviev, A.A. (eds.) Nonlinear Dynamics of the Lithosphere and Earthquake Prediction, pp. 71–139. Springer, Heidelberg (2003)
10. Soloviev, A.A., Maksimov, V.I., Rozenberg, V.L., Ermoliev, Y.M.: Block Models of Lithosphere Dynamics: Approach and Algorithms. In: Wyrzykowski, R., Dongarra, J., Paprzycki, M., Waśniewski, J. (eds.) PPAM 2001. LNCS, vol. 2328, pp. 572–579. Springer, Heidelberg (2002)
11. Wells, D.L., Coppersmith, K.L.: New Empirical Relationships among Magnitude, Rupture Length, Rupture Width, Rupture Area, and Surface Displacement. Bull. Seism. Soc. of America 84(4), 974–1002 (1994)

AMG for Linear Systems in Engine Flow Simulations

Maximilian Emans

AVL List GmbH, Hans-List-Platz 1, 8020 Graz, Austria
maximilian.emans@gmx.at

Abstract. The performance of three fundamentally different AMG solvers for systems of linear equations in CFD simulations using SIMPLE and PISO algorithm is examined. The presented data is discussed with respect to computational aspects of the parallelisation. It indicates that for the compressible subsonic flows considered here basic AMG methods not requiring Krylov acceleration are faster than approaches with more expensive setup as well as recently presented k-cycle methods, but also that these methods will need special treatment for parallel application.

1 Introduction

The three-dimensional simulation of combustion engines requires the approximate solution of the Navier-Stokes equations and an energy equation for unsteady compressible flows in terms of pressure, temperature, and velocity fields. Turbulence treatment, models for combustion processes, formation of NOx and soot e.g., are attached to this kernel. Contemporary simulation tools such as AVL FIRE$^{(R)}$ 2009 provide a considerable amount of freedom with respect to geometry that requires a discretisation on unstructured meshes. Due to the resolution necessary for reasonable modelling, the size of the problems is in the range of one million grid cells or more which makes the use of parallel computers using typically a few CPUs inevitable to keep computing times at an acceptable level.

A common approach to solve the Navier-Stokes equations in this context is the family of algorithms derived from SIMPLE, a general template of algorithms for pressure linked equations. For the solution of the appearing systems of linear equations a fast and sufficiently robust solver is needed. AMG methods are a reasonable option here. However, the development of these methods has not yet ceased and fundamentally new ideas have just been published recently, e.g. by De Sterck et al. [3] and Notay [8]. In this contribution we shall compare the performance of deliberately chosen AMG algorithms applied as linear solvers for systems that appear in the simulation of mainly subsonic flow at the example of a combustion engine. In contrast to related work, e.g. of Čiegis et al. [1] or Starikovičius [12] who have devised a detailed analysis of a single solver algorithm along with a method to predict the parallel performance, we shall merely observe the performance of different algorithms and discuss the opposed effects of increased parallel overhead and decreased memory requirement (per processor) with a growing degree of parallelism onto the effective computing time.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 350–359, 2010.

2 Pressure-Correction Equation

Any algorithm for the solution of the Navier-Stokes equations has to cater for the non-linearity of this system and has to provide a feasible way to couple the equations. Common in commercial tools are the iterative algorithms SIMPLE and PISO, both ensuring the pressure-velocity coupling by the solution of a pressure-correction equation which drives the velocity field towards the condition imposed by the continuity equation through an appropriate correction of the pressure field. For details of the SIMPLE algorithm we refer to Patankar [10], the extension to compressible flows has been provided by Demirdžić [2]; for the PISO algorithm see Issa [6]. PISO can be considered a refined variant of SIMPLE: Where in the correlation between pressure-correction and velocity update of SIMPLE certain terms are simply neglected, the same terms are approximated by an additional iteration in PISO, requiring the solution of linear systems, such that the latter algorithm guarantees continuity within each (outer) PISO iteration. The additional linear systems differ only in the right-hand side.

The most time consuming step of both algorithms is the approximate solution of these pressure-correction equations which have positive definite system matrices for the cases presented here. We know from experience that a reduction of any norm of the residual by a factor between 100 and 10000 is sufficient to allow the SIMPLE or PISO algorithm to converge; this distinguishes our task from the majority of applications of linear equation solvers that are characterised by a much lower residual tolerance.

3 AMG Algorithms

In this section we refer to papers providing detailed descriptions of the algorithms under examination in this contribution and describe our own necessary amendments e.g. for parallelisation.

3.1 General Aspects of Parallel AMG

The fundamental AMG algorithm is not repeated here; for this we refer the reader to the literature, e.g. to the appendix of Trottenberg et al. [13]. We follow the Galerkin approach, see Trottenberg et al. [13], to compute the coarse-grid operator. Given a system matrix A and a restriction operator R we use R^T as interpolation operator and compute the coarse-grid operator A^C explicitly as

$$A^C = RAR^T. \tag{1}$$

Accepting a deterioration of the parallel convergence of the algorithms, we allow only local (with respect to the domain assigned to a certain processor) interpolation to avoid the necessity to transfer parts of the matrix A and to reduce the data transfer during setup. A full parallelisation of the solution phase is also not feasible since the most appropriate smoothers are inherently sequential. The usual practice is to employ fully parallel Jacobi on the boundaries between two domains after the values on the boundaries have been exchanged, and to continue with a Gauß-Seidel scheme for the interior points. This technique is referred to as hybrid Gauß-Seidel smoother, see van Emden and Meier-Yang [4].

3.2 AMG Based on Smoothed Aggregation – ams1cg

This method divides the set of fine-grid points into a number of disjoint subsets that are called aggregates. These aggregates become the coarse-grid points. The tentative interpolation operator with constant interpolation is smoothed by application of one Jacobi step along the paths of the graph of the fine-grid matrix to improve the quality of the interpolation. This serial Smoothed-Aggregation method is described in detail in Vaněk et al. [15]. In the parallel case we do not permit aggregates that range over more than one subdomain and restrict the smoothing to local points since more complex approaches do not seem necessarily to result in better performance, in particular not for low processor numbers, see Tuminaro and Tong [14]. We follow the suggestion of Fujii et al. [5] and start the aggregation process at points adjacent to inter-domain boundaries.

This AMG method is used as a preconditioner for the conjugate gradient algorithm, see Saad [11]. We implemented a v-cycle scheme with two pre- and two post-smoothing hybrid Gauß-Seidel sweeps. In the parallel case grids with less than 200 nodes are merged to one of the neighbours. The equation system of the coarsest grid is solved directly by Gaußian elimination.

3.3 Aggregation-Based AMG with Krylov-Acceleration – amk1fc

This method has recently been suggested by Notay and Vassilevski [8]. To ensure that the computation of the coarse-grid operator is cheap, it splits the set of fine-grid nodes into aggregates of four nodes and uses constant interpolation. For details of the algorithm we refer to Notay [9].

In the solution phase of this algorithm the standard v-cycle is replaced by an adaptive algorithm that approximates the solution of the coarse-grid systems by one or two iterations of a Krylov-subspace method that is recursively preconditioned by itself, see again Notay [9] for details. Due to the adaptive preconditioning the "flexible conjugate gradient" method of Notay [7] (restarted after six iterations) is employed instead of a standard conjugate gradient method. Again two pre- and two post-smoothing hybrid Gauß-Seidel sweeps are performed. The grid hierarchy is complete once one grid has less than 200 cells, while the particular coarse-grid treatment suggested by Notay [9] is not applied since its parameters appeared to be somewhat arbitrary. The equations system of the coarsest grid is solved by a parallel block Gauß-Seidel with four iterations.

3.4 Basic AMG – amggs2

The most obvious application of algebraic (and also geometric) multigrid is its use as a "stand-alone" solver rather than as a preconditioner. As a matter of fact none of the AMG methods described above will show good performance when used as "stand-alone" solver, since convergence is poor. Better results are obtained with methods where the quality of the interpolation is rather high and the computation of the coarse-grid operator is very efficient. The second requirement precludes the application of expensive interpolation methods such that constant interpolation will be the method of choice. Then the first requirement can only

be met if the number fine-grid points per aggregate or C-point is very low, e.g. 2 or 4.

For the coarse-grid selection in our basic AMG we use the algorithm of Notay [9] that produces aggregates comprising not more than two fine-grid nodes since it is documented in an excellent manner. The cycling scheme is an F-cycle, see Trottenberg et al. [13], i.e. recursively a w-cycle is followed by a v-cycle. Only two hybrid Gauß-Seidel sweeps are performed after the return to the finer grid, i.e. no pre-smoothing is done. The coarse-grid treatment is the same as that of the algorithm amk1fc.

4 Computations of Flows in an Engine

Our test cases are taken from the simulation of a full cycle of a gasoline engine. The three-dimensional computational domain is subject to change in time: It contains the interior of the cylinder and the parts of the ducts through which the air is sucked into the cylinder or expelled from it. A three-dimensional simulation of a full engine cycle comprises the simulation of the (compressible) flow of cold air into the cylinder while the piston is moving downward, the subsequent compression after the valves are closed, the combustion of the explosive mixture, and the discharge of the hot gas while the piston moves upward. Since a single simulation run on a parallel computer will still take a few hours computing time, we pick out four short periods of a few time steps, one from each of these four strokes of the cycle. Typical data of the engine cycle is shown in figure 1, the geometry and slices through the meshes can be seen in figure 2.

4.1 Description of the Numerical Experiments

The computationally relevant information about the cases is compiled in table 1 that contains e.g. the memory size of the matrix information, the time step dt,

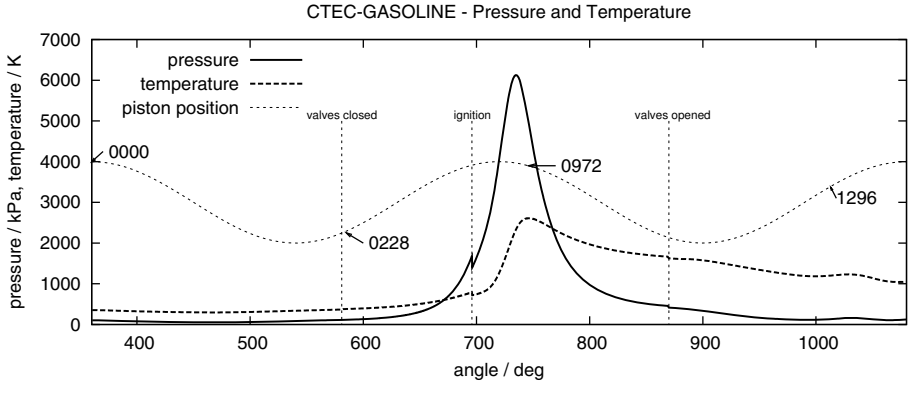


Fig. 1. Scheme of the engine cycle along with notation for the considered partial problems

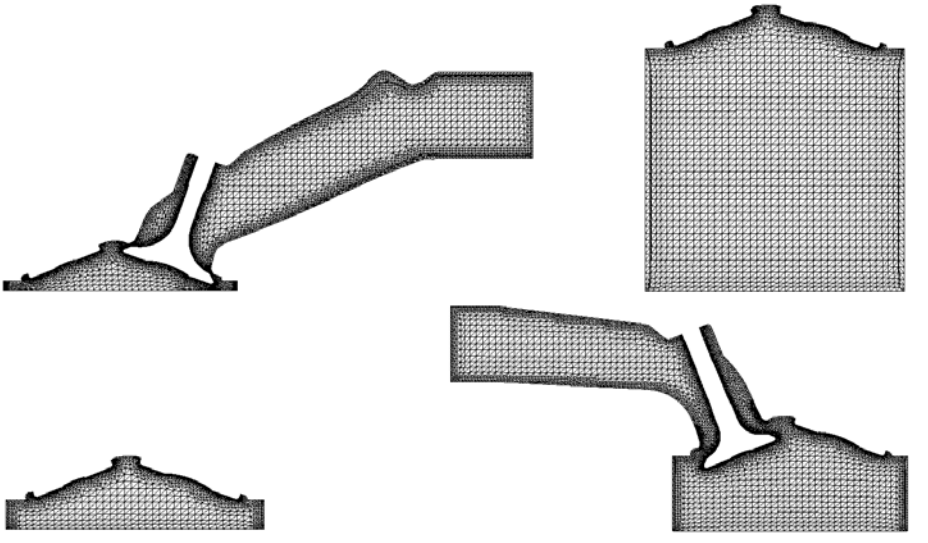


Fig. 2. Slices through the 3-dimensional meshes of the partial problems, from left top clockwise: load, compression, combustion, discharge

Table 1. Characterisation of the cases

no.	stroke	angle	size [MB]	dt $[s]$	n_t	boundaries	n_{sy} SIMPLE	n_{sy} PISO	n_{se} PISO
0000	load	360°	111.0	$3.03 \cdot 10^{-5}$	5	mass flow, wall	130	120	52
0228	compression	585°	23.0	$3.03 \cdot 10^{-5}$	20	wall	172	224	69
0972	combustion	746°	18.6	$6.06 \cdot 10^{-6}$	20	wall	378	512	201
1296	discharge	1013°	52.1	$3.03 \cdot 10^{-5}$	15	pressure, wall	175	130	58

the number of time steps n_t, the number of systems to solve n_{sy} for SIMPLE and PISO and the number of setups n_{se} for PISO (for SIMPLE it equals n_{sy}).

The AMG algorithms described in the previous section are compared to each other and to a reference, a conjugate gradient solver with incomplete Cholesky factorisation as preconditioner, referred to as ichpcg. We ran each case for each number of processors once using each of these algorithms as solver of the pressure-correction equations of the SIMPLE algorithm. All computations were repeated employing PISO instead of SIMPLE. The number of time steps n_t was the same for SIMPLE and PISO, as expected the number of SIMPLE and PISO iterations however was different.

For the measurements we used up to four nodes à 2 quad-cores (i.e. 8 cores) of a Linux-cluster (Intel Xeon CPU X5365, 3.00GHz, main memory 16 GB, L1-cache 2·4·32 kB, L2-cache 2·2·4 MB) connected by an Infiniband (Mellanox) network with an effective bandwidth of approximately 750 Gbit/s. The part of

Table 2. Number of iterations, operator complexity, and number of levels (in brackets) of the computations with the linear AMG solvers

Case	n_p	1	4	16		1	4	16		1	4	16	
		ams1cg			$c/(n_l)$	amk1fc			$c/(n_l)$	amggs2			$c/(n_l)$
0000		416	446	484	1.51	776	733	731	1.57	569	568	564	2.56
0000P		419	447	477	(3)	724	684	683	(7)	538	525	524	(14)
0228		316	348	400	1.52	480	464	464	1.59	419	417	424	2.60
0228P		427	498	534	(3)	673	659	659	(6)	555	554	574	(11)
0972		788	1125	1134	1.50	1414	1376	1313	1.59	1148	1149	1147	2.60
0972P		1534	1664	1817	(3)	2734	2211	1929	(6)	1761	1759	1747	(11)
1296		372	430	455	1.53	570	532	594	1.60	447	451	499	2.62
1296P		308	373	393	(3)	454	444	486	(6)	402	412	447	(12)

the program related to the solver was compiled by Intel-FORTRAN compiler 10.1, the communication is performed through calls to hp-MPI subroutines (C-binding). The test cases were run within the environment of the software AVL FIRE$^{(R)}$ 2009 on 1, 2, 4, 8, and 16 processors, where the domain decomposition was performed once for each case by the standard algorithm provided with this software. Computations with 1, 2, and 4 processors were done on a single node, for 8 and 16 processors we used 2 and 4 nodes respectively such that each processor had full access to 4 MB L2-cache since in preliminary experiments it has been found that the L2-cache is the bottleneck for such kind of computations. Although distributing two or four tasks to two or four nodes would increase the performance, we used a single node for these computations since the gain in performance does usually not justify the occupation of the additional cores in the practical applications.

The raw data of our evaluation is the computing time of the setup that is independent of the number of iterations and the computing time of the solution phase for the SIMPLE and PISO computations, see figures 3 and 4. Furthermore, we present the operator complexity

$$c = \sum_{(l)} \frac{\text{number of matrix elements of level } l}{\text{number of matrix elements of level 1}}, \tag{2}$$

and the cumulative iteration count in table 2. From the measured times the parallel efficiency E_p is computed as $E_p = \frac{t_1}{p \cdot t_p}$, where t_p denotes the computing time on p processors. The values of E_p for SIMPLE and PISO are very similar; for SIMPLE they may be found in figures 3 and 4.

4.2 Performance with SIMPLE Algorithm

The curves are influenced by two main opposed effects. On the one hand, E_p becomes worse as the number of processors is increased. This has three reasons: First the well-known parallel overhead that depends on communication requirements of the algorithm: without further analysis as for example done by Čiegis

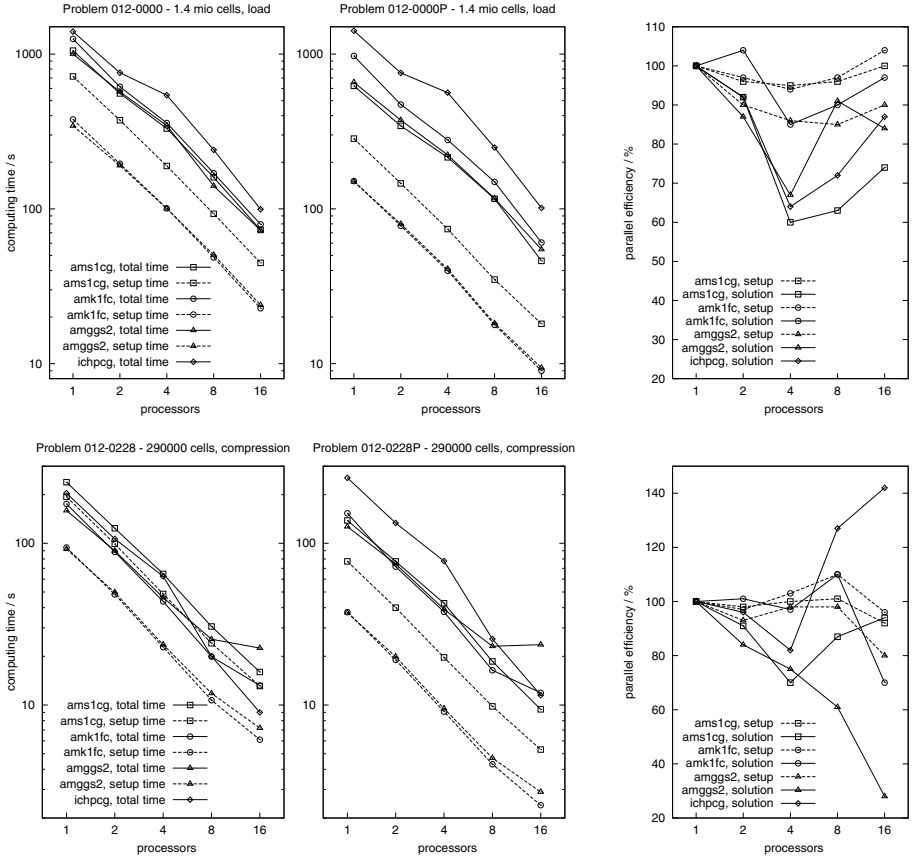


Fig. 3. Computing times for cases 0000 and 0228, using SIMPLE (left) and PISO (middle), and parallel efficiency for solution and setup phase (right) for SIMPLE

et al. [1] this effect is difficult to quantify. However, it depends on the number of exchange operations and is consequently much higher for algorithm amggs2, the algorithm that has the most levels and visits them most frequently due to the F-cycle. Second, certain hardware resources such as memory access are depleted since the computations on up to four processors take place on one node of the cluster; this effect is mainly responsible for the decrease of the parallel efficiency for computations on up to four processors. The third reason is the deterioration of the convergence that is due to imperfect parallelisation; it leads to an increase in the number of iterations of the solver and is independent of the hardware. This effect is essentially only seen for algorithm ams1cg.

The opposed effect is a superlinear acceleration of the computation of the smaller distributed problems due to a decreased probability of cache misses. It is also very hard to predict this effect quantitatively, for a detailed description of a simple test case we refer to Čiegis et al. [1]. Here, the visible consequence is that

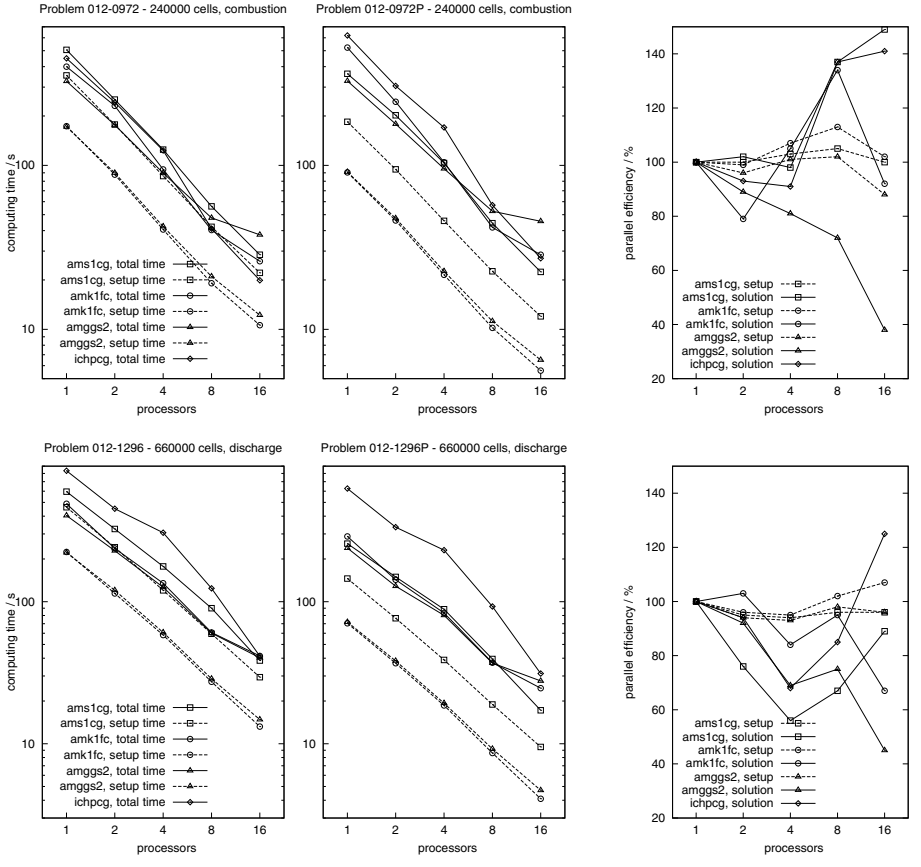


Fig. 4. Computing times for cases 0972 and 1296, using SIMPLE (left) and PISO (middle), and parallel efficiency for solution and setup phase (right) for SIMPLE

the parallel efficiency of certain algorithms rises for computations on more than four cores although one would expect it to decrease due to degraded convergence and additional communication cost. The parallel efficiency may exceed 100% in cases where the gain through cache effects is stronger than the loss through parallelisation. Whereas for algorithm ams1cg the obviously positive and negative effects onto the run-time partly annihilate each other and the positive ones prevail, for algorithm amggs2, characterised by the highest number of levels and the largest memory consumption, only in the largest case (0000) the positive effects dominate over the negative ones.

The significant increase of E_p of amk1fc for two processors is due to the instruction of Notay [9] to use a stricter criterion to determine if a second preconditioning iteration is needed for the parallel case, leading to lower iteration numbers and consequently to lower solution times. The parallel efficiency of amggs2 and amk1fc for 16 processors does not drop in the largest case 0000 the

same way as in the other cases since the ratio of communication cost to cost of other tasks is significantly lower in case 0000; the reason is that this case is much larger than the other cases which entails a larger amount of internal work.

As mentioned, algorithm ams1cg is most vulnerable to the simplification of the parallel setup leading to degradation of the convergence in parallel runs; while this affects the performance only marginally, acceleration through reduction of cache misses and comparatively low impact of communication overhead cater for good parallel efficiency. However, for processor numbers lower than eight this algorithm is slower than the other ones since its setup is expensive.

In all four cases the basic AMG amggs2 is the fastest algorithm for computations on up to four processors, amk1fc comes close in some cases, whereas ams1cg is significantly slower (note the logarithmic scale in figures 3 and 4). For computations on more than four processors the parallelisation reduces the computing time reasonably for ams1cg, but not for amggs2 and amk1fc.

4.3 Performance with PISO Algorithm

While the parallel efficiency is essentially the same as for SIMPLE (and is therefore not shown here) the computing times are different. One observes first that the portion of setup time of the AMG solvers is reduced dramatically as expected since expensively computed coarse-grid hierarchies can be reused several times. As a consequence, ichpcg is significantly slower now than these algorithms for up to eight processors. For the same reason, the difference between ams1cg and amggs2 for up to four processors has been reduced. The performance of the new algorithm amk1fc lies between that of the other two AMG algorithms and ichpcg. Similar as for SIMPLE, amggs2 shows deficiencies at high processor numbers.

5 Conclusions

If this kind of problem is to be solved on less than eight processors, the fastest AMG algorithm of our selection is a basic multigrid method with a very lean setup and without Krylov-acceleration. Even if we apply PISO to permit the reuse of the grid hierarchies, algorithms with expensive setup such as ams1cg are not observed to be significantly quicker. For runs using more than eight processors, or if the coarse-grid hierarchy can be used several times, however, the Smoothed Aggregation algorithm should be preferred due to its good parallel efficiency.

Whereas the Smoothed Aggregation algorithm is slightly affected by the simplifications of the parallelisation, the basic multigrid method suffers from parallel overhead that is due to the high number of levels. It shares this principal problem with the recently presented k-cycle AMG of Notay [9]. Modifications to this kind of algorithms are needed such that they can be used efficiently for parallel computations if it is not possible to reuse the coarse-grid hierarchy. Finally, Smoothed Aggregation AMG is a good choice for modern Linux-clusters whenever the coarse-grid hierarchy can be used more than once.

References

1. Čiegis, R., Iliev, O., Lakdawala, Z.: On Parallel Numerical Algorithms for Simulating Industrial Filtration Problems. Computational Methods in Applied Mathematics 7, 118–134 (2007)
2. Demirdžić, I., Lilek, Ž., Perić, M.: A Collocated Finite Volume Method for Predicting Flows at All Speeds. International Journal for Numerical Methods in Fluids 27, 1029–1050 (1993)
3. De Sterck, H., Falgout, R.D., Nolting, J.W., Meier-Yang, U.: Distance-two Interpolation for Parallel Algebraic Multigrid. Numerical Linear Algebra with Applications 15, 115–139 (2008)
4. van Emden, H., Meier-Yang, U.: BoomerAMG: a Parallel Algebraic Multigrid Solver and Preconditioner. Applied Numerical Mathematics 41, 155–177 (2001)
5. Fujii, A., Nishida, A., Oyanagi, Y.: Evaluation of Parallel Aggregate Creation Orders: Smoothed Aggregation Algebraic Multigrid Method. In: Proc. of the Workshop on High Performance Computational Science and Engineering, HPCSE 2004, Toulouse, France, pp. 99–122 (2004)
6. Issa, R.I.: Solution of the Implicit Discretised Fluid Flow Equations by Operator Splitting. Journal of Computational Physics 62, 45–60 (1985)
7. Notay, Y.: Flexible Conjugate Gradients. SIAM Journal of Scientific Computing 22, 1444–1469 (2000)
8. Notay, Y., Vassilevski, P.S.: Recursive Krylov-based Multigrid Cycles. Numerical Linear Algebra with Applications 15, 473–487 (2008)
9. Notay, Y.: An Aggregation-based Algebraic Multigrid Method, Report No. GANMN 08-02, Service de Métrologie Nucléaire, Université Libre de Bruxelles (2000)
10. Patankar, S.V.: Numerical Heat Transfer and Fluid Flow. Hemisphere Publishing (1980)
11. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. SIAM, Philadelphia (2003)
12. Starikovičius, V., Čiegis, R., Iliev, O., Lakdawala, Z.: A parallel solver for the 3D simulation of flows through oil filters. In: Čiegis, R., Henty, D., Kagstrom, B., Žilinskas, J. (eds.) Parallel Scientific Computing and Optimization. Advances and Applications. Springer Optimization and Its Applications, vol. 27, pp. 181–192 (2009)
13. Trottenberg, U., Oosterlee, C., Schüller, A.: MULTIGRID. Elsevier Academic Press, London (2001)
14. Tuminaro, R.S., Tong, C.: Parallel Smoothed Aggregation Multigrid: Aggregation Strategies on Massively Parallel Machines. In: Proc. of 2000 ACM/IEEE Conference on Supercomputing, Dallas, USA, article no. 5 (2000)
15. Vaněk, P., Brezina, M., Mandel, J.: Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems. Computing 56, 179–196 (1996)

Parallel Implementation of a Steady State Thermal and Hydraulic Analysis of Pipe Networks in OpenMP

Mykhaylo Fedorov

Computer Science Department, West Pomeranian University of Technology, 70210, Zolnierska 49, Szczecin, Poland
mfedorov@wi.ps.pl

Abstract. The considerable computation time of a practical application of sequential algorithms for simulating thermal and flow distribution in pipe networks is the motivating factor to study their parallel implementation. The mathematical model formulated and studied in the paper requires the solution of a set of nonlinear equations, which are solved by the Newton-Raphson method. An object-oriented solver automatically formulates the equations for networks of an arbitrary topology. The hydraulic model that is chosen as a benchmark consists of nodal flows and loop equations. A general decomposition algorithm for analysis of flow and temperature distribution in a pipe network is presented, and results of speedup of its parallel implementation are demonstrated.

Keywords: pipe networks, steady state, flow and thermal analysis, parallel implementation, OpenMP.

1 Introduction

The domain of application of a pipeline network analysis is very wide, e.g. airplane hydraulic, fuel or environmental control systems, district heating systems, air-conditioning systems of buildings, trains or ships, water or gas distribution systems, and so on. The task of a pipeline network system is to convert the magnitudes of parameters fixed at certain boundary system points into the magnitudes prescribed at the other ones. Such parameters are called boundary conditions. The conversion is a result of mutual transformations of different forms of energies. The moving forces of the transformations are finite temperature and pressure differences. To answer the question how such transformations are realized by the system requires solving analysis problem. To do this, we have additionally to fix magnitudes of geometrical parameters (e.g. for tube they are tube diameter, length, roughness, wall thickness, etc.) and boundary conditions (temperatures, pressures). The result of the analysis is a vector of the thermodynamic parameters (pressures, temperatures, enthalpies, etc.) and flow rates. Practical simulations of aircraft environmental control systems [1] demonstrate the dependence of flow on temperature. It follows from the fact that a Reynolds

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 360–369, 2010.

number depends on dynamic viscosity being a function of temperature, in turn. For fluids, change of temperature on 20 K changes the dynamic viscosity almost twice, which can easily be seen from the water property tables [2]. Being a criterion parameter Reynolds' number makes impact on the values of resistances and heat exchange intensity. The impact of a Reynolds number change due to temperature is more essential for laminar or smooth pipe flows [3]. If a pipe network is controllable [4], then any kind of flow (laminar, turbulent) can take place in its various parts. To account the temperature impact, the joint thermal and hydraulic simulation should be performed. Individually, the analysis methods for the simulation of flow distribution and temperatures have been addressed in literature. For the flow steady state analysis the most common methods are those in which independent variables are expressed in terms of the link or chord flows, of the loop flow correction or the nodal heads. In [5], a lack of stability of the nodal method [5,6,7] is reported. In [8,9] the numerical superiority of the flow method over the nodal head method has been proved. The comprehensive analysis, history and examples of the use of these methods are available in [1], [10,11,12,13,14,15]. The methods of the second order are described in [15,16]. In [17] the value of a full-set equation approach is demonstrated which lends itself to the technique of introducing additional equations to describe modified or added network characteristics meeting specified conditions. Being a more general formulation the full-set approach is also employed in this study. The thermal model is a particular form of the first thermodynamic law. The matrix formulation of thermal model is studied, in detail, in [18,19] and, later, in [20]. The general methodology of the thermal-hydraulic simulation is addressed in [21]. The use of it for more and more large networks is becoming more and more time-consuming. It is the motivating factor to study a parallel implementation of the sequential methodology [21]. The OpenMP standard [23] is chosen as a tool for paralleling, because its use is very straightforward.

2 The Mathematical Model

We consider the stationary thermal and flow simulation of incompressible fluid in a pipe network. We do not explicitly take into account pumps in the network, but they could easily be incorporated, as well as the given discharges from the interior nodes. Three basic conservation physical principles are necessary to formulate the model: continuity, momentum and energy. The first one determines a flow rate in the pipe and nodal equations

$$F_i = \sum_{\forall k \in E_i} \dot{m}_k = 0 \quad i = \overline{1, v} \tag{1}$$

implying that at each node i, the sum of flows in pipes incident to i, which numbers belong to the subset E_i is zero. In (1) v is a number of thermal system nodes for which $|E_i| \geq 2$, $\dot{m}_k$ is a flow rate in pipe k. The second principle expresses pressure differences between the pipe section 1 ($P_{k,1}$) and the pipe section 2 ($P_{k,2}$)

$$F_{v+k} = P_{k,1} - P_{k,2} - K_k \left|\dot{m}_k\right| \dot{m}_k = 0 \quad k = \overline{1,e} \ . \tag{2}$$

through flow rates and pipe resistances K_k

$$K_k = \left(L\lambda/2\rho D_h A_c^2\right)_k \ , \tag{3}$$

where D_h- hydraulic diameter, A_c- cross-sectional area, ρ- density, λ - Darcy-Weisbach friction factor, L - pipe length. We use the following formulas for friction factors in this study [24]

$$\lambda = 64/Re \ , \ \ Re \leq 2300, \ \text{laminar flow} \ , \tag{4}$$

$$\lambda = 0.11 \left(\frac{\Delta}{D_h} + \frac{68}{Re}\right)^{0.25} \ , \ \ Re \geq 2300 \ , \ \text{turbulent flow} \ , \tag{5}$$

where Δ is an inner surface pipe roughness, Re is the the Reynolds number

$$Re = \dot{m} D_h / \mu A_c \ , \tag{6}$$

where μ is a dynamic viscosity. The third principle yields two equations

$$\Phi_k = T_{k,i_1} - e^{\left(\frac{UA_s}{Cm}L\right)_k} T_{k,i_2} + \left[e^{\left(\frac{UA_s}{Cm}L\right)_k} - 1\right] T_{k,w} = 0 \ , \ \ k = \overline{1,e} \ , i_1 \neq i_2 \ , \tag{7}$$

$$\Phi_{e+i} = \sum_{\forall k \in E_i} C_k \dot{m}_k T_{k,i} = 0 \ , \ \ i = \overline{1,v} \ , \tag{8}$$

where T_{k,i_1}, T_{k,i_2}, are temperatures at pipe ends incident to nodes i_1 and i_2, $T_{k,w}$ is a boundary temperature (e.g. a tube wall temperature); U is a overall heat transfer coefficient, C is an average heat capacity, A_s is an inner heat transfer area. The overall heat transfer coefficient is obtained from the following formula (see e.g. [2])

$$UA_s = \left[\frac{1}{hA_s} + \frac{D_{h,1} \ln \frac{D_{h,2}}{D_{h,1}}}{k_w A_s}\right]^{-1} \ , \tag{9}$$

which is the integral of the differential equation of heat conduction through a cylindrical wall with the given temperature on the tube outer surface as the boundary condition. In (9), $D_{h,1} \equiv D_h$, but $D_{h,2}$ is an outer tube diameter, h is a heat transfer coefficient, k_w is a thermal conductivity. In turn, equation (7) is a form of the analytical solution of the differential equation of convection heat transfer between incompressible fluid flow in the tube and the inner surface of a tube cylindrical wall (see e.g. [2]). Equation (8) means nodal energy balance. The heat transfer coefficient h in (9) is obtained from the definition of the Nusselt number

$$Nu = hD_h/k \ , \tag{10}$$

which is determined, in turn, as follows [25]

$$Nu = 0.0021 Re^{0.8} Pr^{0.43} (Pr/Pr_w)^{0.14} \ , \ \ Re \geq 10^4 \ , \tag{11}$$

$$Nu = (Nu_l^{6.267}/Nu_t^{5.267})Re^{0.68\ln Nu_t/Nu_l} \ , \ \ 2300 \le Re < 10^4 \ , \tag{12}$$

$$Nu = 1.55(PrReD_h/L)^{1.33}(\mu/\mu_w)^{0.14} \ , \ \ Re < 2300 \ , \tag{13}$$

for turbulent, transitional and laminar flows, respectively. Into (11)-(13) the Prandtl number quantity enters. It is defined as follows

$$Pr = C\mu/k \ , \tag{14}$$

In (12) Nu_l is computed from (13) at $Re = 2300$, and Nu_t is computed from (11) at $Re = 10^4$. All working fluid properties (C, k, μ) are assumed to be functions of mean temperatures (T_m) for every pipe network element. Hence, all the criterial parameters (Re, Nu, Pr) are also functions of temperature. Model (1)-(14) is a nonlinear set of algebraic equations, which is solved with Newton-Raphson method, in this paper. The computer representation of (1) is somewhat different

$$[A_{i,j}]_{v+1,e+v_b}\,[\dot{m}_j]_{e+v_b} = 0 \ . \tag{15}$$

Equation (15) shows that a pseudo-node is introduced to which all the nodes with the given pressures are connected; pseudo-edges v_b in number. In doing so (2) takes the form

$$\underbrace{diag\,[K_{i,j}\ \ |\dot{m}_j|]}_{(e+v_b)\times(e+v_b)}\,[\dot{m}_j]_{e+v_b} = [A_{i,j}]^T_{e+v_b,v+1}\,[P_j]_{v+1} \ . \tag{16}$$

Being generated by a graphic editor, equations (16) represent an undirected graph, at first. To give directions (coordinates) to the graph edges, the depth-first algorithm is used. Equations (16) (its left part) can easily be transformed to the loop form by eliminating the same pressure variables from its right part. Doing so, the right part transforms to a vector of pressure differences at the bounds of pseudo-loops (those having edges being incident to the pseudo-node) or zeros for loops. As a result the final model is a nonlinear set of equations that depends on flow rate and temperature, only.

3 Results and Discussion

From temperature model (7)-(14) analysis it follows that it doesn't have full rank. Consequently, some additional equations must be automatically formed and added to the model [19,21]. To do this, the flow rate distribution must be known. Assuming that it holds, we find out outflow pipes for each node. After mixing each outflow pipe has the same temperature. Then, if $\bar{E}$ is a set of outflow pipes of node i, then their $\left|\bar{E}\right| - 1$ pairs form for each node i a set of equations of the following form.

$$\Phi_{e+i+l} = T_{\bar{E}_i(l)} - T_{\bar{E}_i(l+1)} = 0 \ , \ \ l = \overline{1, n_i - 1} \ , \ \ n_i = \left|\bar{E}_i\right| \ , \tag{17}$$

Together, SLE (7)-(8), (17) has a full rank. With the model at hand, we can go over to answer the question how many times faster we can solve (3)-(17) in

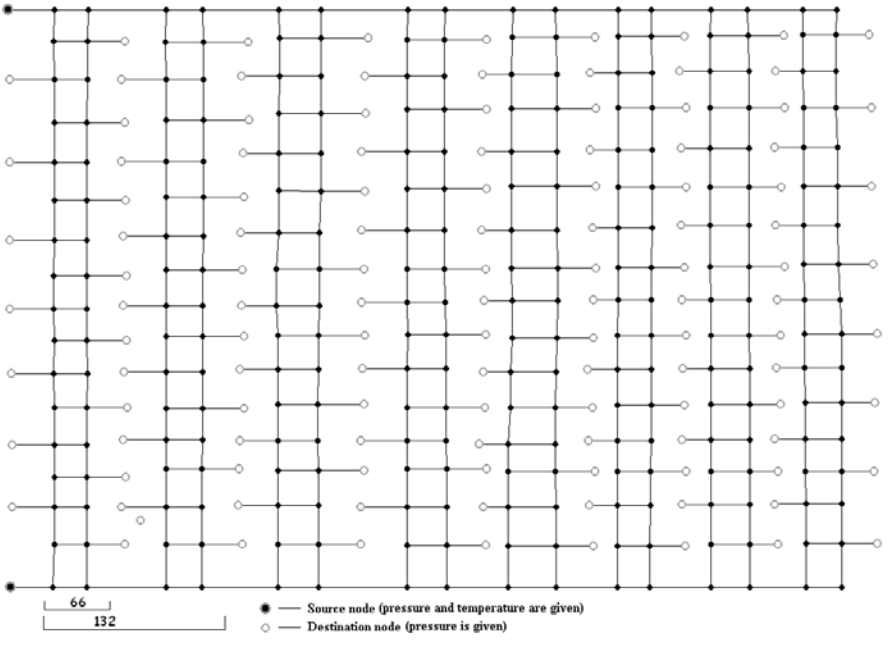


Fig. 1. The benchmark pipe network

parallel than sequentially. With this end in view, a number of pipe networks configurations followed from the benchmark network (see Fig. 1) have been solved sequentially and in parallel. The configurations of pipe networks with the pipe number less than 66 are not depicted. Adding incrementally blocks of 66 pipes the maximum pipe number networks that has been solved has 528 pipes. Numerical experiments have been performed on the computer with 2×Quad Core Processors (Intel Xeon E5405 Quad Core Processor) under the Windows Server 2003 operation system. A pipe network solver is developed and compiled in the Visual Studio Team 2008 environment. The parallel implementation of the sequential algorithm is coded with the OpenMP standard, which substantially simplifies studies on parallelization, because programming with this standard is very straightforward [23]. Experience shows [1,12,21,26] that a general solution algorithm of the nonlinear model described in section 2 can be decomposed into several stages (see the pseudo-code below): hydraulic, thermal and working fluid property ones. Stage 1 operates at fixed mean temperatures. The hydraulic model computes flow rates at constant resistances with Newton's method [27], followed by parallel computation of new resistances being functions of flow rates. As a component of Newton's method, the parallel sparse LU solver is implemented that uses Crout-like reduction with row pivoting. At each Newton iteration, the LU solver solves the following set of linear equations (SLE)

$$J(\dot{m}_i)\Delta\dot{m}_i = -F(\dot{m}_i) \ , \tag{18}$$

obtained from linearizing (15)-(16) and transforming (16) to the loop form, for the full-step flow correction vector, $\Delta\dot{m}_i$.

```
function ṁ = hydraulic_model() { \\ the fine-grained parallelization is
\\ implemented for all loops in the below functions
  while(δK > ε1) { \\ resistance fixed-point iterations
    while(δṁ > ε) { \\ Newton iterations for flow model analysis
      form_hydlaulic_model_SLE(in K, in ṁ, out J, out F);
      ṁ^new =solve_SLE_in_parallel(in J, in F); \\ it solves (18)
      ṁ^new =line_search(in ṁ^new);
      δṁ =estimate_relative_error(in ṁ^new, in ṁ);
      ṁ = ṁ^new ;
    }
    #pragma omp for \\ independent loop iterations
    for(i=0;i<number_of_pipes;i++)
      K^new = compute_resistances(in ṁ); \\ evaluates (5) or (6)
    δK = estimate_relative_error(in K^new, in K);
  }
}
function thermal-hydraulic_model() { \\ a mean temperature is given
  while(δT_m > ε2) { \\ mean temperature fixed-point iterations
    \\ the first stage
    ṁ = hydraulic_model();
    \\ the second stage. B is a right part of (7), (8), (17)
    form_temperature_model_SLE(in C, in ṁ, out Φ, out B);
    T =solve_SLE_in_parallel(in Φ, in B);
    \\ the third stage
    #pragma omp for \\ independent loop iterations
    for(i=0;i<number_of_pipes;i++) {
      compute_mean_temperatures(in T,out T_m^new) ;
      compute_fluid_properties_and_model_parameters(in T_m^new);
    }
    δT_m = estimate_relative_error(in T_m^new, in T_m);
  }
}
```

To assure decreasing $\|F(\dot{m})\|_2^2/2$, a line search technique [27] is employed to correct $\Delta\dot{m}_i$, if necessary. Stage 2 operates both at fixed flow rates and mean temperatures (see pseudocode). In this case, (7)-(8), (17) constitute an SLE with constant coefficients (9)-(14). Stage 3 computes average temperatures and fluid properties. Each iterative algorithm of the general one solves for one type of parameter vector and requires about 6-9 iterations to converge with accuracy $\epsilon \leq 10^{-9}$, $\epsilon \leq 10^{-6}$, $\epsilon_2 \leq 10^{-3}$ for flow rates, resistances and mean temperatures, respectively. One of the basic activities of the algorithm is to solve SLE. Figure 2 depicts matrix portraits of flow and thermal equations of the pipeline

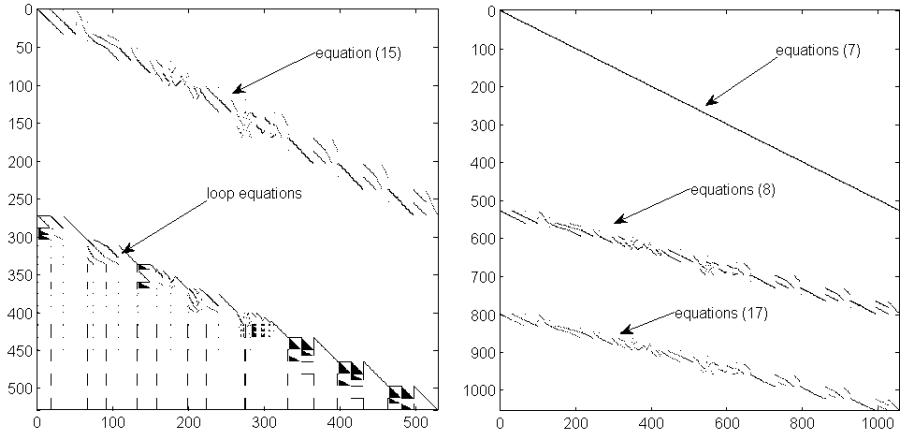


Fig. 2. Matrix portrets of flow (left) and thermal (right) models

network shown in Fig. 1. The SLEs are unsymmetrical and sparse that requires implementation of the solution algorithm for this general case of SLE. Numerical experiments have shown that the speedup of solving SLE benchmarks [28] of size of order 1000 by the LU factorization method doesn't exceed 2, for the given hardware (which configuration can be essential to achieving high parallel performance [22]). Therefore, we can expect that the speedup of the whole algorithm will be of the similar order. Newton's method is largely sequential in nature. It consists of the sequence of subtasks such as formulation equations, symbolic factorization, solving SLE, computing gradients and norms, etc., with numerous conditional constructs, reduction variables and a set of loops having small number of operations. Paralleling each task individually we realize fine-grained parallelization that requires the current task to be synchronized before running the next one that, in turn, might cause essential synchronization overheads. Despite the fact, numerical experiments verify (Fig. 3) scalability of fine-grained parallelism. Indeed, we can observe stable, though not intensive, growth in speedup and efficiency starting even with relatively small pipe network sizes (about 150 pipes). The next numerical experiments that are made give insight in the speedup effects of applying fixed-point iterations (the model decomposition) to the algorithm virsus pipe network sizes and types of the models being solved, and that may be individually commercially useful. The aim is to verify growth in speedup with increase in element models complexity. Curve 1 in Fig. 4 demonstrates the speedup of parallelization of the flow model analysis, where resistances K are kept fixed (see the pseudocode). We can note, that speedup is observed for networks sizes being greater than 250 pipes (see Fig. 4). Experiments with the hydraulic model (curve 2) and the complete model (curve 3) demonstrate that speedup substantially increases. In comparison to the flow model the speedup observed is for minimal synchronization overheads incurred with the parallel inmplementation of the relaxation method, which correct

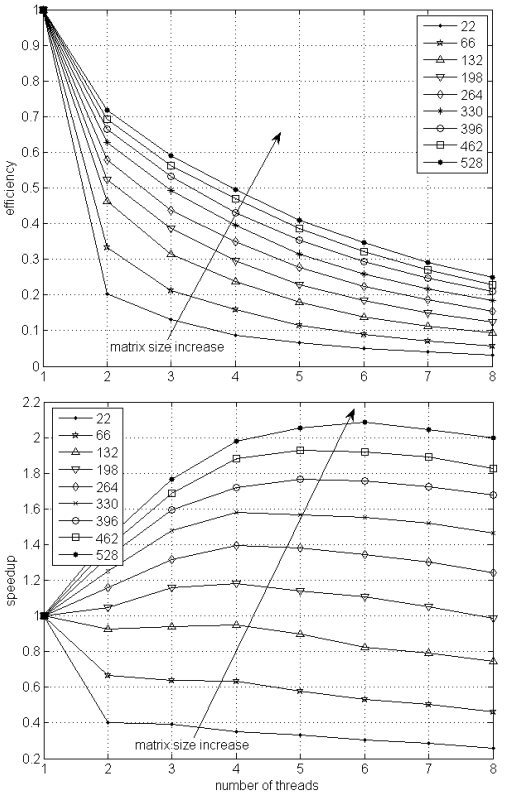


Fig. 3. The speedup and efficiency

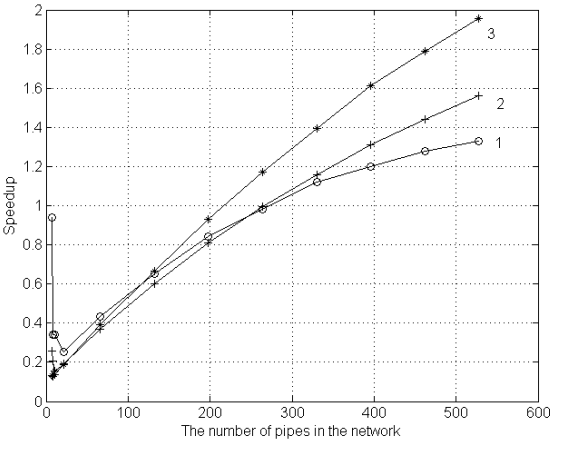


Fig. 4. Speedup of models corresponding different algorithm stages

resistances (stage 1), and average temperatures and working fluid properties (stage 3). Speedup will increase even greater if elements model (e.g. heat exchanger) of a pipe network require more computations for evaluating resistances and mean temperatures.

4 Conclusions

In the paper, the parallel implementation of the steady-state thermal-hydraulic analysis in OpenMP is presented. The mathematical model studied contains all typical tasks inherent to such an application domain. To demonstrate the influence of each task on the final speedup numerical experiments have been carried out for different sets of subtasks and sizes of pipe networks on computer with 2×Quad Core Processors (Intel Xeon E5405 Quad Core Processor) under the Windows Server 2003 operation system. The object-oriented C++ code of the thermal-hydraulic analysis solver, which proved to be robust and scalable, has been implemented and compiled in Visual Studio Team 2008 environment. The paper demonstrates that the fine-grained parallel implementation in OpenMP of the decomposition algorithm considered results in speedup of order 2 for the network of 528 pipes, and the potential in speedup increase exists for larger sizes of pipe networks. It has been achieved by parallelization of the algorithm for solving SLE, being a dominant functionality of the flow model anaysis and temperature submodels. It has been shown that SLE parallelization contribution to the overall speedup grows slowly when a pipe network size increases, while two fixed-point iteration procedures that compose the submodels into the whole are the major sources of the earned speedup value. It is clear that the model can be easily extended to simulate pipe networks with more complex models of element (e.g. with heat exchangers) or working fluid properties (e.g. with account for phase transitions); doing so the speedup will always grow.

References

1. Kondrashenko, V., Vinnichuk, S., Fedorov, M.: Simulation of Gas and Liquid Distributing Systems. Naukova Dumka, Kiev (1990) (in Russian)
2. Massoud, M.: Engineering Thermofluids. Springer, Heidelberg (2005)
3. Idelchik, I.E.: Handbook of Hydraulic Resistances. Machynostrojenije, Moscow (1992) (in Russian)
4. Fedorov, M.: On Software Design of Thermal Systems Steady State Control. Polish J. Environm. Stud. 4C, 279–283 (2008)
5. Donachie, R.P.: Digital Program for Water Network Analysis. J. Hydraulics Div. 100, 393–403 (1974)
6. Chandrashekar, M.: Extended Set of Components in Pipe Networks. J. Hydraulic Div. 106, 133–149 (1980)
7. Nogueira, A.C.: Steady-State Fluid Network Analysis. J. Hydraulic Eng. 119(3), 431–436 (1993)
8. Nielsen, B.N.: Methods for Analyzing Pipe Networks. J. Hydraulic Eng. 115(2), 139–157 (1989)

9. Altman, T., Boulos, P.F.: Convergence of Newton Method in Nonlinear Network Analysis. Mathl. Comput. Modelling 21(4), 35–41 (1995)
10. Chandrashekar, M.: Extended Set of Components in Pipe Networks. J. Hydraulic Div. 106, 133–149 (1980)
11. Evdokimov, A.G.: Optimal Problems of Engineering Networks. Wyzsha Shkola, Charkov (1976) (in Russian)
12. Merenkov, A.P., Chasilev, V.J.: Theory of Hydraulic Circuits. Nauka, Moscow (1985) (in Russian)
13. Larock, B.E., Jeppson, R.W., Watters, G.Z.: Hydraulics of Pipeline Systems. CRC Press, Boca Raton (2000)
14. Osiadacz, A.J.: Steady-state Simulation of Gas Networks. Fluid systems Sp., Warszawa (2001) (in Polish)
15. Sennikova, E.V., Sidler, V.G.: Mathematical Simulation and Optimization of Evolving Heat Supply Systems. Nauka, Novosibisk (1987) (in Russian)
16. Stevanovic, V.D., Prica, S., Maslovaric, B., Zivkovic, B., Nikodijevic, S.: Efficient Numerical Method for District Heating System Hydraulics. Energy Conversion & Management 48, 1536–1543 (2007)
17. Boulos, P.F., Wood, D.J.: Explicit Calculation of Pipe Network Parameters. J. Hydraulic Eng. 116, 1329–1344 (1990a)
18. Fedorov, M.: Automation of Mathematical Model Construction for the Analysis of Heat Regimes of Heat Exchanger Systems by the Method of Gauss Convolution. Electronic Modeling 22(6), 19–25 (2000) (in Russian); Engineering Simulation vol.18, pp. 727–734 (2001) (English translation)
19. Fedorov, M.: Steady-State Simulation of Heat Exchanger Networks. Electronic Modeling 24(1), 101–111 (2002) (in Russian)
20. Filho, L.O.F., Queiroz, E.M., Costa, A.L.H.: A Matrix Approach for Steady-State Simulation of Heat Exchanger Networks. Applied Thermal Eng. 27, 2385–2393 (2007)
21. Fedorov, M.: Thermal Modes Simulation of Heat Exchanger Networks Having Turbo-machines. In: Proceedings of the Int. Conf. on Marine Technology IV, pp. 309–314. WIT Press, Southampton (2001)
22. Čiegis, R., Čiegis, R., Meilūnas, M., Jankevičiūtė, G., Starikovičius, V.: Parallel mumerical algorithm for optimization of electrical cables. Mathematical Modelling and Analysis 13(4), 471–482 (2008)
23. OpenMP Standard, http://openmp.org
24. Altschul, A.D.: Hydraulic Resistances. Nedra, Moscow (1970) (in Russian)
25. Kanevets, G.E.: Heat Exchangers and Heat Exchanger Systems. Naukowa Dumka, Kiev (1982) (in Russian)
26. Bialecki, R.A., Kruczek, T.: Frictional Diathermal Flow of Steam in a Pipeline. Chem. Eng. Sc. 51(19), 4369–4378 (1996)
27. Dennis, J., Schnabel, R.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. SIAM, Philadelphia (1996)
28. Rice, J.R.: Matrix computations and mathematical software. McGraw-Hill, New York (1996)

High-Performance Ocean Color Monte Carlo Simulation in the Geo-info Project

Tamito Kajiyama[1], Davide D'Alimonte[2],
José C. Cunha[1], and Giuseppe Zibordi[3]

[1] CITI, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Quinta da Torre, 2829–516 Caparica, Portugal
t.kajiyama@di.fct.unl.pt, jcc@di.fct.unl.pt

[2] CENTRIA, Departamento de Informática, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Quinta da Torre, 2829–516 Caparica, Portugal
davide.dalimonte@gmail.com

[3] Global Environment Monitoring Unit, Joint Research Centre, 21027 Ispra, Italy
giuseppe.zibordi@jrc.it

Abstract. The Geo-Info project aims to provide Geoscience experts with software toolkits tailored for selected target application domains. Within this framework, high-performance computing (HPC) solutions are here applied to optimize a Monte Carlo (MC) code to support Ocean Color (OC) investigations. The present paper introduces the Geo-Info project, describes the HPC solutions applied for the OC MC case study, and gives early performance results focusing on runtime, speedup, and parallel efficiency.

1 Introduction

Scientific computing for Geosciences (e.g., Earth, Ocean and Space sciences) increasingly depends on high-performance computing (HPC) resources because of computation time and memory requirements. This poses various Computer Science issues whose solution requires a tailored support by means of HPC techniques. Nowadays computer architectures are highly complex because parallelism and heterogeneity are ubiquitous. Achieving high-performance on the modern computer hardware tends to require sophisticated parallelization and optimization, while relevant technologies rapidly change both in terms of hardware and software aspects. Therefore, a close collaboration between researchers in Geosciences and Computer Science is of particular importance to make best use of HPC resources for real-world problem solving.

Aiming at promoting interdisciplinary research in Geosciences and Computer Science, a grant-in-aid project named Geo-Info has been established. The project, funded by the Portuguese Ministry of Science, Technology and Higher Education (MCTES), intends to facilitate joint efforts among the Centre of Informatics and Information Technology (CITI), Centre of Artificial Intelligence (CENTRIA), Centre of Geological Science and Engineering (CICEGe), and Centre of Oceanography (CO), the first three belonging to the Universidade Nova de Lisboa and the last to the Universidade de Lisboa.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 370–379, 2010.

The general objective of the Geo-Info project is to provide domain experts with software toolkits tailored for selected target application domains in Geosciences. The project aims at addressing a diversity of issues including programming support for the development of domain-specific models and algorithms, packaging of the models and algorithms into reusable software components, workflow specifications by means of the software components, as well as execution management of the software components dynamically mapped onto distinct parallel and distributed computing environments.

The case study presented here supports scientific experimentation in the Ocean Color (OC) application domain. Specifically, HPC solutions are designed for optimizing the execution time of a Monte Carlo (MC) code developed to model the radiative transfer processes in seawater [1,2] and analyze uncertainties in radiometric products derived from in-water optical profiles. The MC code traces the trajectories of a large number of photons that undergo random events of scattering and absorption in the seawater medium. The accuracy of OC MC simulations depends on the photon population size, which directly translates into computation time. Hence, OC MC simulations are clearly a prime example of computationally highly demanding applications that require specifically geared HPC support.

This paper is organized as follows. Section 2 overviews the HPC framework for Geosciences. Section 3 presents MC simulations of radiative transfer processes within the context of OC applications. Finally, Section 4 presents preliminary performance results obtained in two different parallel computing environments.

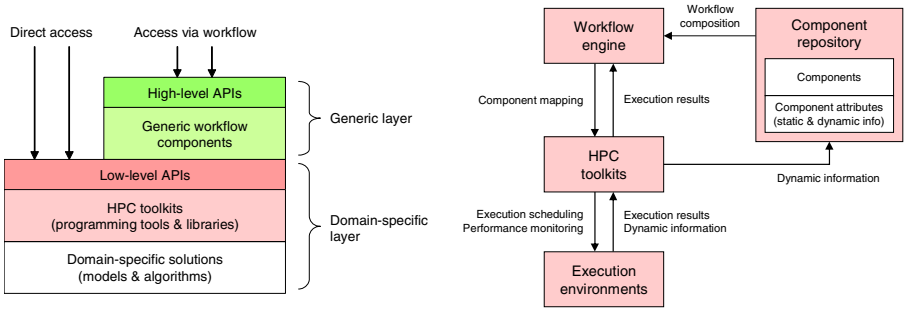
2 The HPC Framework for Geosciences

This section presents the HPC framework under development in the Geo-Info project. The HPC framework is composed of two major layers, the domain-specific layer and the generic layer, as shown in Fig. 1 (a) and described in the following paragraphs.

The domain-specific layer consists of 1) domain-specific solutions, 2) HPC toolkits, and 3) low-level application program interfaces (APIs). The domain-specific solutions comprise application-oriented scientific models and algorithms, implemented and executed based on HPC toolkits offered by the HPC framework. The HPC toolkits provide domain scientists with an easy-to-use suite of programming tools and libraries specifically designed for selected target application domains in Geosciences. The HPC toolkits also give access to the domain-specific solutions through the low-level APIs.

The generic layer provides a high-level interface to access the domain-specific solutions. These solutions are packaged into reusable software components so that they can be executed through high-level APIs using scientific workflow engines.

Users can apply domain-specific solutions in two different ways: 1) directly through the low-level APIs (useful for application developers to locally execute



(a) Layered architecture. (b) Basic building blocks.

Fig. 1. Schematic overview of the HPC framework for Geosciences

and debug the HPC applications being implemented); and 2) through the high-level APIs using the workflow engines (useful for domain experts to conduct experiments by running the reusable software components remotely).

Figure 1 (b) shows four basic building blocks of the HPC framework. A dedicated workflow engine is built (based on extensions to existing workflow engines [3,4]) to meet scientific needs of the target applications, as well as special requirements for high-performance computing. The workflow engine enables, in terms of a graphical language, a composition of workflow that describes the data flow among processing tasks. Note that processing tasks are execution units in the workflow engine and their enactment involves the invocation of reusable software components.

A component repository is meant to supply software components to the workflow engine together with component attributes describing static and dynamic aspects of each component. The component attributes are used to dynamically map each software component onto an appropriate execution environment.

HPC toolkits reside between the workflow engine and execution environments, managing the execution of software components. The HPC toolkits are also intended to offer various runtime services, for instance execution scheduling and performance monitoring. Dynamic information, such as the performance of software components on a particular execution environment, is gathered and stored in the component repository. Information of this kind constitutes the dynamic component attributes to tune the mapping of software components onto execution environments at runtime.

3 Monte Carlo Simulations for Ocean Color Applications

Monte Carlo simulations of radiative transfer processes in the Ocean Color application domain were selected as a case study for a joint effort of oceanographers and computer scientists in the context of the Geo-Info project. The scope of the simulations is to support the understanding of uncertainties affecting data products derived from in-water optical profiles.

3.1 Ocean Color

Ocean color remote sensing allows for retrieving the concentration and optical properties of the so-called *optically significant seawater components*, which affect the in-water light field [1]. Remote sensing OC products are derived applying bio-optical algorithms to reflectance spectra measured by space-born sensors. Chlorophyll-a concentration, generally used as a proxy for biomass concentration, is the most known and utilized OC product. It is exploited in climate change studies [5] and has potential use to water quality monitoring [6].

Accurate *in situ* radiometric measurements are fundamental for the validation of remote sensing (RS) primary products (i.e., RS radiometric data corrected for the atmospheric perturbations) and for the development of schemes for the determination of higher level RS products (i.e., the concentration and properties of optically significant seawater components). Within such a framework, an uncertainty of at most 5% is the common target for *in situ* radiometric measurements [7] applied to OC calibration and validation activities.

The present study relies on MC simulations of radiative transfer processes to investigate perturbations introduced by the wave effects and sensor tilt, which are commonly neglected in uncertainty budgets of *in situ* radiometric products. Results are expected to contribute to a more thorough application of *in situ* measurements within current OC space missions (i.e., MERIS on-board of the ENVISAT platform and MODIS on-board of the AQUA platform).

3.2 Field Measurements

In situ radiometric products can be derived from in-water optical profiles taken with free-fall systems (Fig. 2). Radiometric quantities of interest are the upwelling radiance L_u, as well as the upward and downward irradiance E_u and E_d [8,9]. It is recalled that L_u is the spectral radiant energy at nadir view per unit time, unit area and solid angle, and generally expressed in units of $\mathrm{mW\,cm^{-2}\,\mu m^{-1}\,sr^{-1}}$. The quantities E_u and E_d are the spectral radiant energy incident per unit time and unit area upon a horizontal surface. These are generally expressed in units of $\mathrm{mW\,cm^{-2}\,\mu m^{-1}}$.

The light intensity decreases exponentially as a function of depth in a water layer with homogeneous optical properties [1]. Omitting for brevity the wavelength dependence and indicating with $\mathfrak{R}$ the radiometric quantities defined above (i.e., L_u, E_u or E_d), the value of $\mathfrak{R}(z)$ at depth z can then be expressed in terms of its subsurface value $\mathfrak{R}(0^-)$ and the diffuse attenuation coefficient $K_{\mathfrak{R}}$ as

$$\mathfrak{R}(z) = \mathfrak{R}(0^-)\mathrm{e}^{-K_{\mathfrak{R}}\cdot z}.$$

In practice, given a series of N measurements of $\mathfrak{R}$ at various depths z_i in a layer close to the sea surface ($z_1 < z_i < z_2$, see Panel (c) in Fig. 2), the subsurface value $\mathfrak{R}(0^-)$ and the slope $K_{\mathfrak{R}}$ are determined from the least square regression of log-transformed $\mathfrak{R}(z_i)$ as a function of z_i.

Other *in situ* radiometric products, in addition to the subsurface values and the diffuse attenuation coefficients, are the *irradiance reflectance* R and the *normalized water leaving radiance* L_{WN} computed as

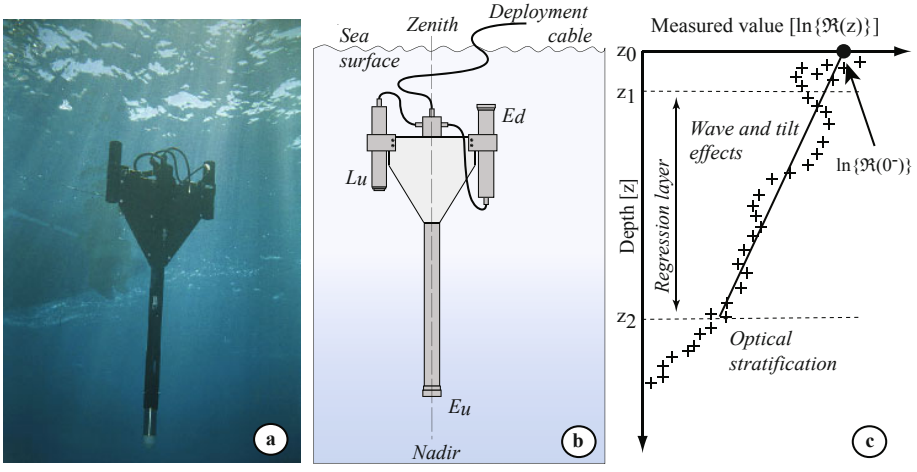


Fig. 2. Panel (a) shows an example of a free-fall profile system used to measure the in-water light distribution (Courtesy of Scott McClean, Satlantic Inc., Halifax). The positions of the sensors to measure the up-welling radiance (L_u), and the upward and downward irradiance (E_u and E_d, respectively) are highlighted in Panel (b). The joint effect of the surface wave focusing and the instrument tilt is schematically shown in the radiometric profile of Panel (c).

$$R = E_u(0^-)/E_d(0^-)$$

and

$$L_{\mathrm{WN}} = 0.54 \cdot \overline{F}_0 \cdot L_u(0^-)/E_d(0^+),$$

where $\overline{F}_0$ is the mean extraterrestrial solar irradiance, 0.54 is the transmittance for the up-welling radiance from below to above the sea surface and $E_d(0^+)$ is the above water downward irradiance. The normalized water leaving radiance is the reference quantity for validating remote sensing data products.

3.3 Monte Carlo Simulations

The overall uncertainty budget of *in situ* radiometric products depends on various sources. These include uncertainties in the absolute calibration, sensitivity change of radiometers responsivity between successive calibrations, environmental effects and measurement perturbations. Among the various sources of uncertainty, the least explored and quantified are those due to environmental perturbations like wave effects, and those induced by variations of the asset (i.e., tilt) of radiometers during profiling. Wave effects mostly result in light focusing and defocusing by surface wave facets which alter the light distribution in the water column [10,11]. Radiometer tilt (i.e., inclination of the sensor longitudinal axis with respect to the ideal zenith-nadir direction) produces artifacts in profile data. The instrument tilt recorded during the radiometer deployment can be used to restrict the data analysis to those measurements performed with tilt

below a threshold (i.e., typically 5 degrees). However, once the threshold has been applied, the residual tilt effects are convoluted with wave effects, which depend on the radiometer deployment speed and the characteristics of the waves at the sea surface. As a matter of fact, the accurate quantification of individual uncertainties due to wave and tilt effects cannot be easily addressed experimentally [11]. This limitation can be overcome through theoretical investigations with MC simulations of in-water radiometric fields.

By restricting simulations to inelastic processes, the interactions of light with the molecules and particles constituting the seawater medium are described through absorption and scattering. In this study, MC simulations of photon transport are addressed as follows [1,2]:

1. The photon travel distance is derived from the optical path-length l sampled from an exponential probability density function $p(l) = \exp(-l)$ where $l > 0$. Assuming homogeneous inherent optical properties, the geometrical travel distance s is computed as $s = l/c$, where c is the attenuation coefficient.
2. The probabilities that a photon undergoes an absorption or scattering process are a/c and $1 - a/c$, respectively (with a indicating the absorption coefficient).
3. The photon scattering angle θ is here sampled from the Henyey-Greenstein scattering phase function $p_{HG} = 0.5 \cdot (1 - g^2) / \left[(1 - 2\cos\theta + g^2)^{3/2}\right]$, where g describes the degree of anisotropy [12].

The present MC application has been designed to generate two-dimensional representations of L_u, E_u and E_d. These representations are used to produce *virtual* in-water radiometric profiles. As in the real world, virtual profiles obtained with the same illumination conditions and water volume optical properties might differ from each other due to the random effects of tilt and wave perturbations. MC simulations offer then the unique opportunity to compare profile data and derived radiometric products, with and without accounting for the variability introduced by the wave and tilt perturbations. The demonstration case illustrated in this study shows the applicability of the method without comprehensively compiling the uncertainty budget for tilt and wave perturbations under different environmental conditions.

4 Preliminary Performance Results

Based on a sequential MATLAB code implementing the MC algorithm described in the previous section, a parallel OC MC code in C was developed with the aims of 1) providing oceanographers with a handy tool to accelerate experimentation in the OC domain; and 2) identifying domain-specific issues to be addressed with the HPC framework in the Geo-Info project.

MC simulations of radiative transfer processes belong to the class of embarrassingly parallel problems, since transport of each photon can be computed independently of the others. We developed 2 parallel versions of the MC code using MPI and OpenMP. In both versions, the total number of photons P is

Table 1. Performance of the MPI-based MC code on a heterogeneous AMD Opteron cluster. The number of photons in each test is 10^6.

#procs.	Time [sec.]	Speedup	Efficiency
1	448.6	1.00	100.0%
2	267.7	1.68	83.8%
4	131.6	3.41	85.2%
8	66.02	6.80	84.9%
12	44.72	—	87.4%

evenly divided into N processors. Each processor computes the trajectories of P/N photons using a different random number seed and generates 3 matrices to record two-dimensional representations of L_u, E_u and E_d. For each radiometric quantity, matrices on the N processors are gathered and summed up at the end of the simulation. A matrix summation requires $N-1$ matrix additions, so that the total number of matrix additions is $3N-3$.

The parallel MC code was tested in 2 different cluster environments. Since the MPI and OpenMP versions of the MC code showed similar performance, only the results with the MPI version are presented.

4.1 Small Heterogeneous Cluster Case

Performance results of the MPI-based code on a small PC cluster are shown in Table 1. The cluster consists of 4 compute nodes, where 2 nodes have 4-way AMD Opteron 2.2 GHz processors and the other 2 nodes have 2-way AMD Opteron 2.0 GHz processors (the total number of cores is 12). The nodes are interconnected by Infiniband. The cluster runs Scientific Linux 5.0, and LAM/MPI 7.1.2 and GCC 4.1.2 (with `-O3 -march=opteron` options) were used. The number of traced photons is $P = 10^6$.

Only the 2.2 GHz processors were used to perform simulations with $N \leq 8$. In these homogeneous cases, speedup S and parallel efficiency E are defined as $S = T(1)/T(N)$ and $E = S/N$, where $T(1)$ and $T(N)$ are the execution times with 1 and N processors, respectively, in units of seconds.

The classical definitions of speedup and efficiency are no longer applicable in the heterogeneous case with $N = 12$. In this case, the parallel efficiency was evaluated by applying the definition of efficiency by Colombet *et al.* [13]. Suppose that $u_1\%$ of the photons are assigned to each of the eight 2.2 GHz processors and $u_2\%$ of the photons to each of the four 2.0 GHz processors. Similarly, $v_1\%$ and $v_2\%$ of $3N-3$ matrix additions are assigned to each 2.2 GHz and 2.0 GHz processor, respectively. Assessed average tracing time per photon is $U_1 = 4.49 \times 10^{-4}$ sec. on a 2.2 GHz processor and $U_2 = 5.15 \times 10^{-4}$ sec. on a 2.0 GHz processor, while a matrix addition takes $V_1 = 1.37 \times 10^{-2}$ sec. on a 2.2 GHz processor and $V_2 = 1.56 \times 10^{-2}$ sec. on a 2.0 GHz processor. Now, consider the following linear programming problem:

$$\text{Minimize } T \text{ subject to } \begin{cases} U_1 P u_1 + V_1(3N-3)v_1 = T \\ U_2 P u_2 + V_2(3N-3)v_2 = T \\ 8u_1 + 4u_2 = 1 \\ 8v_1 + 4v_2 = 1 \\ u_1 \geq 0, u_2 \geq 0, v_1 \geq 0, v_2 \geq 0 \end{cases}$$

The solution T of this problem is a lower bound of the parallel execution time and thus can be considered as the ideal execution time T_{id}. Parallel efficiency on heterogeneous processors is then defined as $E = T_{\text{id}}/T(N)$. In the case of $P = 10^6$, the result is $T_{\text{id}} = 39.1$ sec. (with $u_1 = 8.72\%$, $u_2 = 7.57\%$, $v_1 = 0\%$, $v_2 = 25.0\%$) and $E = 87.4\%$. Although the MPI-based MC code was tested without any load balancing mechanism, the analysis above shows that the parallelization strategy of evenly distributing the photons to the 12 heterogeneous processors was indeed quite efficient.

4.2 Large-Scale Homogeneous Cluster Case

Figure 3 shows performance results of the MPI-based code on TACC Ranger (University of Texas at Austin). Ranger consists of 3936 SMP compute nodes interconnected with Infiniband. Each node has 4-way quad-core AMD Opteron 2.3 GHz processors and 32 GB memory, running a Linux OS. The code was built with MVAPICH 1.0.1 and Intel C Compiler 10.1 (with `-O3 -xW -ipo` options). The number of traced photons is 4×10^6 in all cases with different numbers of processors. The MC code showed a linear scalability up to 2048 cores. The serial execution time was 1424.0 sec., whereas the parallel execution time with 2048 processors was 1.6572 sec. (i.e., $S = 859.30$ and $E = 42.0\%$). The relatively low efficiency with the largest number of processors is attributed to the the number of photons per processor, which was very small in this experiment, as well as to the increase in the $O(\log \log N)$ time spent for parallel matrix summations.

Figure 4 illustrates 3 output matrices populated with the trajectories of 10^9 photons. The execution time was 1771.5 sec. using 256 cores on Ranger. The large number of photons was intended to reduce MC intrinsic noise in the L_u distribution, since the L_u sensor has a narrow field-of-view angle (20 degrees in this simulation) and only a relatively small fraction of photons are recorded

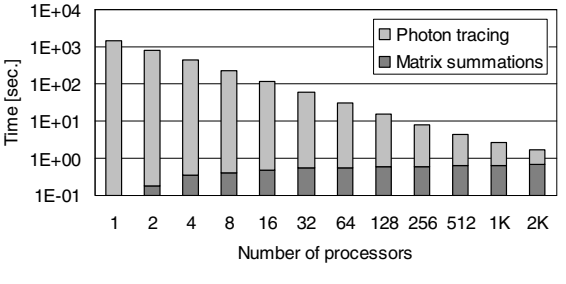


Fig. 3. Performance of the MPI-based MC code on TACC Ranger. The number of traced photons in each test is 4×10^6.

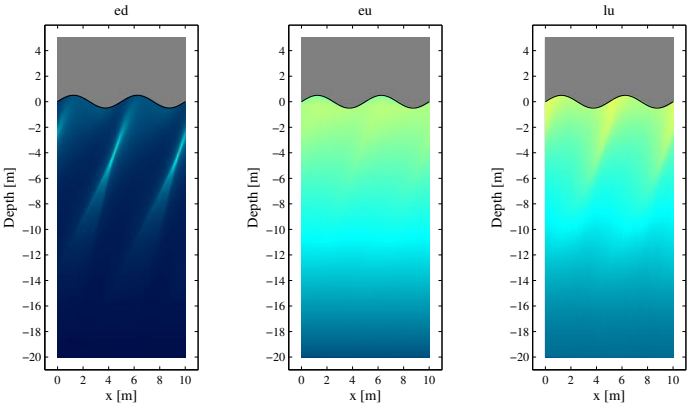


Fig. 4. Examples of simulated light distribution. The number of traced photons is 10^9.

into the corresponding output matrix when compared with E_u and E_d. This exemplifies the computationally demanding nature of OC MC simulations, thus motivating the use of large-scale parallel computing environments like Ranger.

5 Concluding Remarks

This paper presented an HPC framework for Geosciences developed in the GeoInfo project. A case study to support radiative transfer simulations for OC applications was conducted to collect preliminary performance results of a parallel MC code. It is remarked that it would be cumbersome for OC domain scientists to use classical programming and execution strategies to gather, in a reasonable amount of time, the statistical figures needed to investigate the tilt and wave focusing effects with MC simulations. On the contrary, the performance results from the pilot investigation discussed here have shown that the HPC approach ultimately opens the venue – both in terms of computational efficiency and application transparency – to additional investigations aiming at a more comprehensive exploitation of *in situ* radiometric measurements for the calibration of space sensors and the validation of remote sensing data products.

Our future work includes the development of HPC solutions based on hybrid application of message passing (MPI) and shared memory (OpenMP) models to run OC MC simulations on a wider range of heterogeneous parallel and distributed computing environments, as well as their performance evaluation with realistic simulation settings. Outcomes are expected to enable the exploration of large-scale solutions to further increase the accuracy of MC simulations.

Acknowledgments. We would like to thank Dr. Daniel Stanzione and the staff of the Texas Advanced Computing Center (TACC) for their help with large-scale experiments on Ranger in the framework of the Advanced Computing initiative

of the UT Austin-Portugal Program, funded by the Portuguese Foundation for Science and Technology (FCT).

References

1. Mobley, C.D.: Light and Water: Radiative Transfer in Natural Waters. Academic Press, London (1994)
2. Leathers, R.A., Downes, T.V., Davis, C.O., Mobley, C.D.: Monte Carlo radiative transfer simulations for ocean optics: A practical guide. Technical report, Naval Research Laboratory, NRL/MR/5660–04–8819, Washington, DC (September 2004)
3. Altintas, I., Berkley, C., Jaeger, E., Jones, M., Ludäscher, B., Mock, S.: Kepler: An extensible system for design and execution of scientific workflows. In: Proc. 16th Intl. Conf. on Scientific and Statistical Database Management (SSDBM 2004), pp. 21–23 (June 2004)
4. Taylor, I., Shields, M., Wang, I., Harrison, A.: Visual Grid workflow in Triana. Journal of Grid Computing 3(3-4), 153–169 (2005)
5. Sarmiento, J.L., Slater, R., Barber, R., Bopp, L., Doney, S.C., Hirst, A.C., Kleypas, J., Matear, R., Mikolajewicz, U., Monfray, P., Soldatov, V., Spall, S.A., Stouffer, R.: Response of ocean ecosystems to climate warming. Global Biogeochemical Cycles 18 (2004)
6. Hu, C., Chen, Z., Clayton, T.D., Swarzenski, P., Brock, J.C., Muller-Karger, F.E.: Assessment of estuarine water-quality indicators using MODIS medium-resolution bands: Initial results from Tampa Bay, FL. Remote Sensing of Environment 93(3), 423–441 (2004)
7. McClain, C.R., Feldman, G.C., Hooker, S.B.: An overview of the SeaWiFS project and strategies for producing a climate research quality global ocean bio-optical time series. Deep Sea Research Part II: Topical Studies in Oceanography 51(1-3), 5–42 (2004); Views of Ocean Processes from the Sea-viewing Wide Field-of-view Sensor (SeaWiFS) Mission: Volume 1
8. Zibordi, G., Berthon, J.F., Doyle, J.P., Grossi, S., van der Linde, D., Targa, C., Alberotanza, L.: Coastal Atmosphere and Sea Time Series (COASTS), Part 1: A Tower-Based, Long-Term Measurement Program, NASA Goddard Space Flight Center, SeaWiFS Postlaunch Technical Report Series, TM–2002–206892, Greenbelt, MD, June 2002, vol. 19, pp. 1–29 (2002)
9. D'Alimonte, D., Zibordi, G.: The JRC Data Processing System. NASA Goddard Space Flight Center, SeaWiFS Postlaunch Technical Report Series, TM–2001–206892 Greenbelt, MD, vol. 15, pp. 52–56 (May 2001)
10. Zaneveld, J.R., Boss, E., Hwang, P.: The influence of coherent waves on the remotely sensed reflectance. Optics Express 9(6), 260–266 (2001)
11. Zibordi, G., D'Alimonte, D., Berthon, J.F.: An evaluation of depth resolution requirements for optical profiling in coastal waters. Journal of Atmospheric and Oceanic Technology 21(7), 1059–1073 (2004)
12. Henyey, L., Greenstein, J.: Diffuse radiation in the galaxy. Astrophysical Journal 93, 70–83 (1941)
13. Colombet, L., Desbat, L.: Speedup and efficiency of large-size applications on heterogeneous networks. Theoretical Computer Science 196, 31–44 (1998)

EULAG Model for Multiscale Flows – Towards the Petascale Generation of Mesoscale Numerical Weather Prediction

Zbigniew P. Piotrowski, Marcin J. Kurowski,
Bogdan Rosa, and Michal Z. Ziemianski

Institute of Meteorology and Water Management,
ul. Podlesna 61, 01-673 Warsaw, Poland
Zbigniew.Piotrowski@imgw.pl
http://www.imgw.pl/

Abstract. EULAG is an established, highly parallel model for simulating fluid flows across a wide range of scales. It is known to scale well up to 16000 processors on IBM Blue Gene/W. It is noteworthy for its non-oscillatory integration algorithm MPDATA, advanced elliptic solver and generalized coordinate formulation. In this paper we focus on complex orographic flows and present the perspective of implementing EULAG as a high resolution weather prediction tool for Europe.

As resolution of numerical models improves, numerical weather prediction enters the phase where traditional convection parameterization becomes obsolete and is replaced with a cloud-resolving approach. The boundary conditions, especially the topography, are becoming more and more complicated, demanding higher accuracy and better conservation properties from the numerical model construction. This calls for seeking fast and precise fluid solvers.

We present preliminary results of simulations of the flow over realistic topography of the Alps, which proves the model capability to handle steep slopes. We demonstrate performance of the code on IBM Blue Gene/L architecture and compare different I/O strategies, from simple one-node operations to MPI I/O solution. An example of application of VAPOR, a tool for visualization and analysis of tera-scale sized data sets is provided.

1 Introduction

During the recent 50 years, numerical weather prediction (NWP) became the basic tool used for operational weather forecasting. NWP is based on numerical solution of physical equations describing atmospheric flows and processes, for given initial conditions based on observations and measurements.

Atmospheric flows are characterized by spatial scales ranging from global (10^7 m) to micro (about 1 m) for the boundary layer and even much less (10^{-5} m) for microphysical processes in clouds. The NWP requires, on one hand, possibly high spatial resolution to adequately describe the essential physical processes

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 380–387, 2010.

influencing weather and, on the other hand, a solution time significantly shorter than the time of physical realization of the process itself.

In consequence, during the recent decades, the operational NWP employs the highest obtainable computer resources and a nested technique with basically two classes of models. They are: global, solved for the whole planetary atmosphere, and mesoscale, solved for limited areas with higher resolutions and boundary conditions provided by the global ones. The spatial resolution of operational NWP models results from a compromise between the two requirements, defined above, and increases with growing computer capabilities. It currently reaches about 50 km for global models, and 5 to 2 km for the mesoscale ones. Atmospheric processes with scales unresolved by the models have to be parameterized by appropriate procedures, mimicking their statistical effects on resolved parameters.

Currently, with the spatial resolution of operational mesoscale NWP models approaching 1 km scale, they become able, at least partly, to represent explicitly convective processes, responsible for important classes of weather events, especially the severe ones (strong showers leading to flush-floods, thunderstorms, strong wind bursts, etc). On the other hand, the dynamical cores of the current mesoscale models encounter significant problems, as robust numerical stability is required for representing strong convective flows and steep orography.

Academic communities and national weather services of many countries continue research and development work on NWP models. Some of the services, especially in Europe, established scientific consortia, to join their efforts. Polish Institute of Meteorology and Water Management (IMGW) is a member of the two international consortia of mesoscale modeling: ALADIN and COSMO. IMGW established a new scientific team to participate in a work of consortium COSMO on a development of a robust dynamical NWP core, applicable for resolutions of 1 km, and beyond, in the frame of a "Conservative Dynamical Core" project. IMGW proposes to base the research on a dynamical core of the EULAG model, developed in National Center for Atmospheric Research (NCAR), USA, as the scientific multi-scale model for general flows and to test its applicability for NWP purposes.

The task to adopt EULAG for use in NWP is challenging and includes numerous changes to the model physics, code structure and interfaces. In this paper, the authors present the first steps in the adaptation process, namely experiments with flow past steep orography, code scaling test with parallel Tau profiler, 3D visualization using Vapor package, as well as give an example of new MPI/IO module application with relevant performance information.

2 Methodology

Governing equations of EULAG model (1) (see also www.mmm.ucar.edu/eulag) defined in an anelastic approach have the expanded form (2):

$$\frac{D\mathbf{v}}{Dt} = -\nabla\left(\frac{p'}{\bar{\rho}}\right) + \mathbf{g}\frac{\theta'}{\bar{\theta}} - \mathbf{f}\times\mathbf{v}' + \mathcal{M}' + \mathbf{D}, \tag{1}$$

$$\frac{D\theta}{Dt} = D_\theta, \tag{2}$$

$$\nabla \cdot (\bar{\rho}\mathbf{v}) = 0, \tag{3}$$

where $\mathbf{v}$, θ and p are the velocity vector, potential temperature and pressure respectively, D/Dt denotes material derivative. All primed variables are deviations from the ambient (environmental) state, $\bar{\rho}$ is an ambient density profile, $\mathcal{M}'$ symbolizes appropriate metric forces due to coordinate transformations, $\mathbf{D}$ and D_θ are diffusion terms for kinematic viscosity and thermal diffusivity, $\mathbf{f}$ and $\mathbf{g}$ are Coriolis and gravity parameters respectively.

Each prognostic equation of any fluid variable ψ (e.g. temperature, velocity, water vapour, cloud condensate, precipitation mixing ratio, etc.), written as:

$$\frac{\partial \psi}{\partial t} + \nabla \cdot (\mathbf{v}\psi) = R, \tag{4}$$

where R combines all forcings and/or sources, is discretized with a use of non-oscillatory forward-in-time (NFT), 2nd order (in time and space) scheme:

$$\psi_i^{n+1} = \mathcal{LE}_i(\psi^n + \frac{1}{2}\Delta t \mathbf{F}^n) + \frac{1}{2}\Delta t \mathbf{F}_i^{n+1}. \tag{5}$$

Here, $\mathcal{LE}$ represents either an advective semi-Lagrangian or a flux-form Eulerian transport operator (MPDATA, e.g. (3)). Indices i and n denote spatial and temporal location on a Cartesian mesh, respectively, and Δt is the time step of the model. The Eulerian algorithm employs point-wise integrals of evolution equations while semi-Lagrangian one is based upon the trajectory-wise approach. Equation (5) represents an implicit system with respect to the dependent variable ψ. Completion of the model algorithm requires a straightforward algebraic inversion of equation (5), resulting in the formulation of the boundary value problem for pressure implied by the mass continuity constraint (3). The resulting elliptic equations are solved, subject to appropriate boundary conditions, using the preconditioned generalized conjugate-residual approach, a non-symmetric Krylov-subspace solver; cf. (4) and references therein for comprehensive discussions. The model has several preconditioning options. Here, we used two iterations (per each iteration of the Krylov solver) of an implicit stationary Richardson scheme, in the spirit of an ADI or block-Jacobi preconditioner; see (4) for further details.

3 Results

With refining resolution of mesoscale NWP models, complexity of topography becomes an increasingly important challenge for forecasting codes. Integrating flow along steep slopes can result in numerical instability of model solution. Smoothing the orography is sometimes applied to ensure the stability, however, such simplification modifies the physical problem and puts in question correctness of the result.

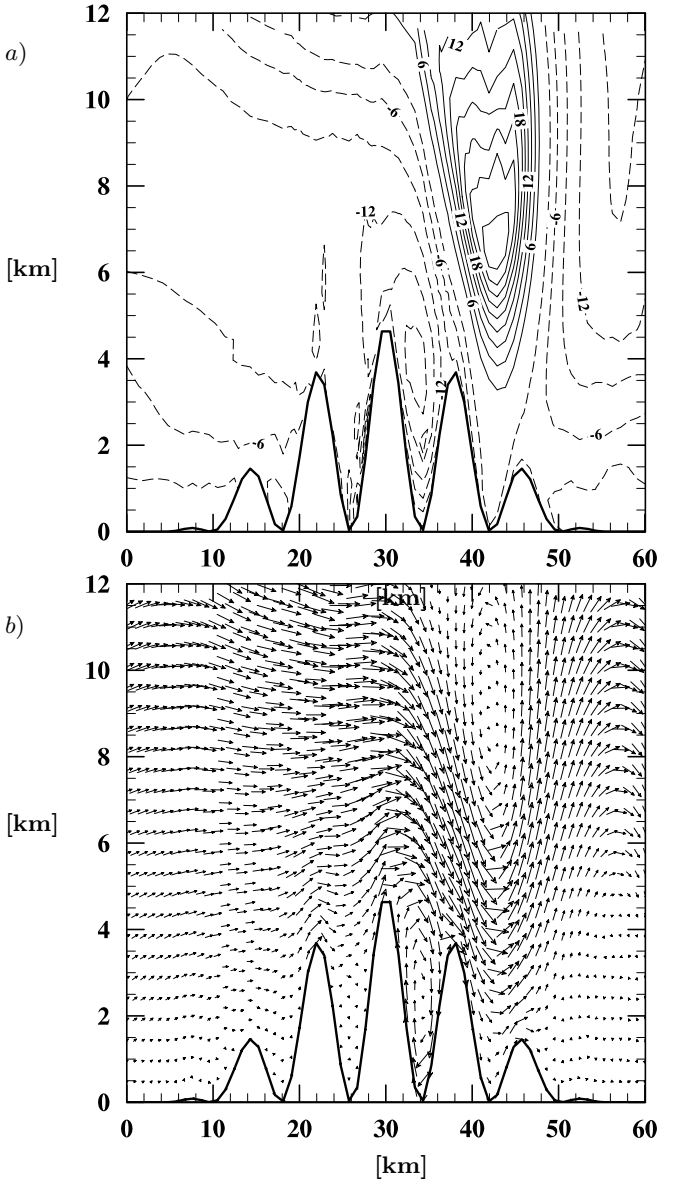


Fig. 1. 2D flow over the undulating terrain. Potential temperature perturbation distribution *a*) and corresponding velocity flow field *b*) obtained using the EULAG model. The picture illustrates solution after about 7 hours of simulated time. The environmental horizontal component of velocity is equal 30 m/s.

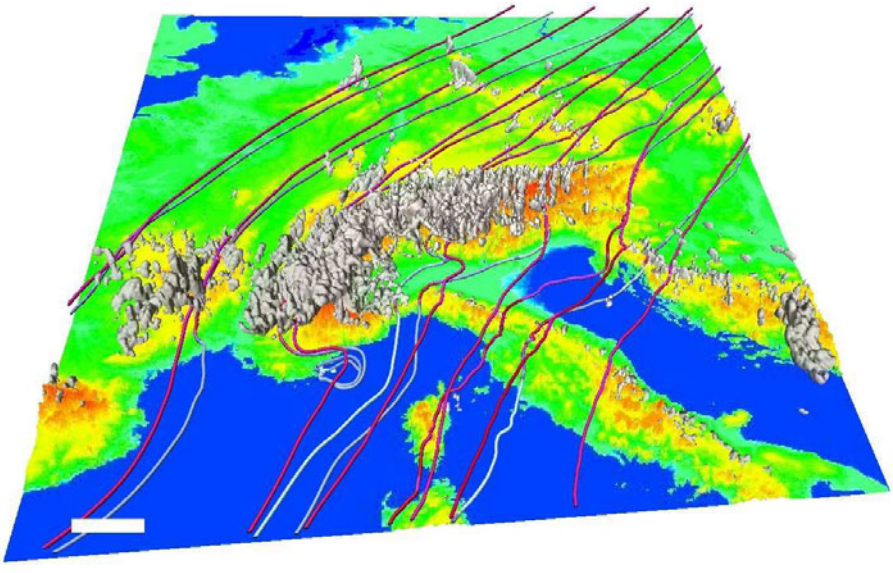


Fig. 2. Orographically forced convection over the Alps after 12 h of constant (7 m/s) environmental inflow from NE direction. Grey isosurfaces depict cloud regions while the streamlines indicate the instantaneous direction of the wind flow between 0-3 km height with bright lines at lower levels to dark lines in the upper layer.

We present results of experiments performed using EULAG model, which suggest that the code is capable to preserve numerical stability even for the flows over strongly undulated topography. Two different cases, i.e. two- and three-dimensional (2D and 3D) are analyzed. The first simulation corresponds to the experiment described by Schär et al. (5). In order to imitate the effects of complex topography we employ the mountain with a finescale structure. Several tests with different wind speed and different amplitude of the hills were performed. The example result for a case of strong wind (30 m/s) and elevated hill (5 km) is shown in Fig. 1. We observe local circulations resolved between steep slopes of the valleys.

More realistic scenario of the three dimensional test alludes the NWP formulation. Uniform environmental windflow about 7 m/s from the North-East direction is simulated over Central and Western Europe. Topographic data are based on an unfiltered MeteoSwiss orography. Resolution of the domain is $700 \times 700 \times 121$ nodes and the unstretched grid box size (i.e., before terrain-following transformation of coordinates) is about 2.2 km $\times$ 2.2 km $\times$ 125 m, which reproduces a typical setup of current advanced operational NWP models. Ambient relative humidity was constant in the whole domain and was set to 0.82. To emphasize orographic effects, we dismiss radiative processes and surface fluxes. Subgrid scale processes are represented by implicit LES property of MPDATA scheme (see (6) for recent application and references therein). Clouds are

forming solely due to orographically induced vertical air currents. Such phenomenon is typically responsible for moist processes (condensation, evaporation and precipitation) in mountain regions. In our model the basic bulk approach of moist thermodynamic has been employed (7). Figure 2 displays the volumes of clouds and instantaneous direction of the air flow (streamlines) after 12 h of the real time integration (wall clock time of the whole run is about 5 h on 500 two-processor nodes of IBM Blue Gene/L - BGL machine).

Figure 2 has been produced with Vapor[1] (8), an advanced graphical 3D tool.

4 Model Performance

Several tests of EULAG code performance has been completed using parallel profiler from Tau[2] package on NCAR IBM Blue Gene/L system with 1024 compute nodes (2048 processors). In the first test we compare the speedup of the code for runs with different number of processors. The speedup is defined as the ratio $T_{N_{\min}}/T_N$ where $T_{N_{\min}}$ corresponds to the total time of the run with the smallest number of processors (here $N_{\min}$=140). T_N is the duration of runs with larger number of CPUs (here $N = 250, 500, 1000$). For testing purpose we used 3D setup described in the previous paragraph, complexity of which is somewhat representative to the NWP model. Figure 3 presents almost linear scalability of the code up to 1000 CPUs. The results are close to an ideal scaling and for 1000 CPUs the difference is merely $\sim 10\%$. Additional scaling tests were performed in

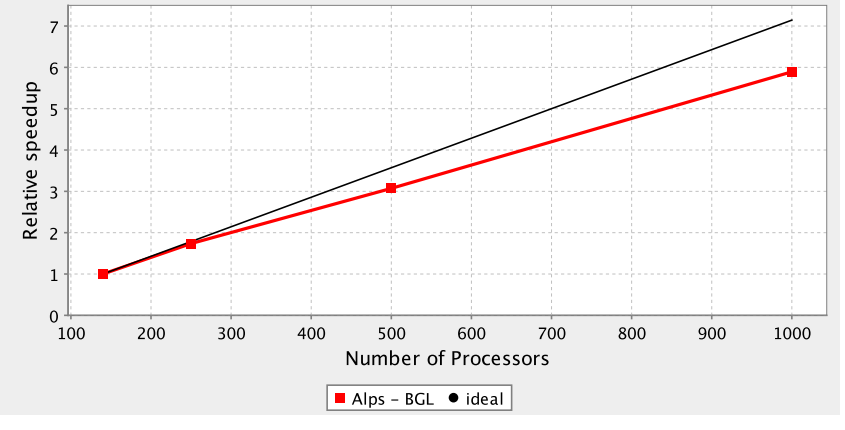


Fig. 3. Strong scaling results

[1] Vapor is a modern visualization and analysis platform designed for terascale datasets, capable for displaying volume rendered data, isosurfaces, probes of 3D data, streamlines and trajectories. It may be coupled with IDL interpretive data language. (www.vapor.ucar.edu).

[2] TAU Performance System is a portable profiling and tracing toolkit for performance analysis of parallel programs (www.cs.uoregon.edu/research/tau).

Table 1. Time in seconds spent for saving output data of the size $\sim 9\ GB$

$\#CPUs$	Sequential	Parallel
250	196.8	48.3
500	172.4	36.7
1000	158.3	32.0

order to evaluate the time needed to save model variables set to a disk space. For use in NWP, data needs to be stored in one of the recognized data formats. Historically, EULAG model allowed only sequential Fortran tape dump. Recently implemented parallel method of writing data to disc storage is based on MPI-I/O technology provided by Parallel Netcdf, an extension to popular meteorological data format Netcdf. Table 1 contains the information about the time spent for saving the same amount of data (full set of model variables, here $\sim 9\ GB$) with different number of CPUs. Parallel method of data saving is about 4 times (for 250 CPUs) and almost 5 times (for 1000 CPUs) faster then sequential one. For extensive discussion of EULAG's scaling on different architectures, see (9).

5 Summary

We have presented EULAG, an established computational model for simulating flows, set in a scenario of future weather prediction models with horizontal resolution of order O(1 km). We have demonstrated model capability to resolve flows past steep and complicated idealized and realistic mountainous terrain. We have performed, using parallel profiler, the scaling tests of realistic 3D flow simulation, which proved good scalability of the model. Implementing MPI I/O solution with Parallel Netcdf package has shown to reduce I/O overhead several times. Construction of the next-generation NWP prediction model is a challenging task. EULAG model reveals to be a promising fluid solver for weather forecasting applications with an excellent tolerance to the complicated topography. This paper confirms and expands earlier results of very good scalability of the model, as well as shows its significant improvement due to parallelisation of its I/O procedures.

Acknowledgments

Computer time was provided by NSF MRI Grant CNS-0421498, NSF MRI Grant CNS-0420873, NSF MRI Grant CNS-0420985, NSF sponsorship of the National Center for Atmospheric Research, the University of Colorado, and a grant from the IBM Shared University Research (SUR) program and Academic Computer Centre in Gdansk (TASK).

References

[1] Smolarkiewicz, P.K., Margolin, L.G.: On forward-in-time differencing for fluids: An Eulerian/semi-Lagrangian nonhydrostatic model for stratified flows. Atmos.-Ocean Special 35, 127–152 (1997)

[2] Lipps, F.B., Helmer, R.S.: A scale analysis of deep moist convection and some related numerical calculations. J. Atmos. Sci. 39, 2192–2210 (1982)
[3] Smolarkiewicz, P.K., Margolin, L.G.: MPDATA: A finite-difference solver for geophysical flows. J. Comput. Phys. 140, 459–480 (1998)
[4] Smolarkiewicz, P.K., Temperton, C., Thomas, S.J., Wyszogrodzki, A.A.: Spectral preconditioners for nonhydrostatic atmospheric models: extreme applications. In: Proceedings of the ECMWF Seminar Series on Recent developments in numerical methods for atmospheric and ocean modelling, Reading, UK, September 6-10, pp. 202–220 (2004)
[5] Schär, C., Leuenberger, D., Führer, O., Luthi, D., Girard, C.: A new terrain-following vertical coordinate formulation for atmospheric prediction models. Mon. Weather Rev. 130, 2459–2480 (2002)
[6] Piotrowski, Z.P., Smolarkiewicz, P.K., Malinowski, S.P., Wyszogrodzki, A.A.: On numerical realizability of thermal convection. J. Comput. Phys. 228, 6268–6290 (2009)
[7] Grabowski, W.W., Smolarkiewicz, P.K.: A multiscale anelastic model for meteorological research. Mon. Weather Rev. 130, 939–956 (2002)
[8] Clyne, J., Mininni, P., Norton, A., Rast, M.: Interactive desktop analysis of high resolution simulations: application to turbulent plume dynamics and current sheet formation. New J. Phys. 9(8), 301 (2007)
[9] Prusa, J.M., Smolarkiewicz, P.K., Wyszogrodzki, A.A.: EULAG, a computational model for multiscale flows. Comp. Fluids 37(9), 1193–1207 (2008)

Parallel Implementation of Particle Tracking and Collision in a Turbulent Flow

Bogdan Rosa[1] and Lian-Ping Wang[2]

[1] Institute of Meteorology and Water Management,
ul. Podlesna 61, 01-673 Warsaw, Poland
bogdan.rosa@imgw.pl
[2] Department of Mechanical Engineering, 126 Spencer Laboratory,
University of Delaware, Newark, Delaware 19716-3140, USA
lwang@udel.edu
http://www.me.udel.edu/∼lwang/

Abstract. Parallel algorithms for particle tracking are central to the modeling of a wide range of physical processes including cloud formation, spray combustion, flows of ash from wildfires and reactions in nuclear systems. Here we focus on tracking the motion of cloud droplets with radii in the range from 10 to 60 μm that are suspended in a turbulent flow field. The gravity and droplet inertia are simultaneously considered. Our codes for turbulent flow and droplet motion are fully parallelized in MPI (message passing interface), allowing efficient computation of dynamic and kinematic properties of a polydisperse suspension with more than 10^7 droplets. Previous direct numerical simulations (DNS) of turbulent collision, due to their numerical complexity, are typically limited to small Taylor microscale flow Reynolds numbers (~ 100), or equivalently to a small physical domain size at a given flow dissipation rate in a turbulent cloud. The difficulty lies in the necessity to treat simultaneously a field representation of the turbulent flow and free movement of particles. We demonstrate here how the particle tracking and collision can be handled within the framework of a specific domain decomposition. Our newly developed MPI code can be run on computers with distributed memory and as such can take full advantage of available computational resources. We discuss scalability of five major computational tasks in our code: collision detection, advancing particle position, fluid velocity interpolation at particle location, implementation of the periodic boundary condition, using up to 128 CPUs. In most tested cases we achieved parallel efficiency above 100 %, due to a reduction in effective memory usage. Finally, our MPI results of pair statistics are validated against a previous OpenMP implementation.

1 Introduction

Air turbulence plays an important role in warm rain development. The process has been extensively investigated in many scientific studies ([1]; [2]; [3]), but complete quantitative understanding is still insufficient. The general description of the multiscale interaction of cloud droplets with turbulent air flow is a challenging task due

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 388–397, 2010.

to inherent nonlinearities, inhomogeneities and coupling over disparate length and time scales. One of the main tools being used for cloud microphysics study is direct numerical simulation (DNS). Numerical complexity limits the DNS of turbulent collision to small Taylor microscale Reynolds number (R_λ ∼100). In real clouds the value is a few orders of magnitude larger ($R_\lambda \sim 10^4$). Achieving such high Reynolds numbers in numerical simulation is not feasible, but fortunately small flow structures are mainly responsible for droplets collision thus a truncated representation of turbulence is useful. Here, we intend to bring the numerical modeling closer to the physical conditions by extending the size of the computational domain. Serial codes or codes parallelized in OpenMP (e.g. (4)) for particle tracking can be run only on computers with shared memory where computational resources are limited to a single node. This restricts the grid resolution typically below 256^3. To perform simulation of turbulent collision on 512^3 grid or higher, different method of parallelization has to be applied.

In this paper, we report on the development of DNS of turbulent collision with parallel MPI library so the resulting code can be run on platforms with a distributed memory. Such a treatment allows the use of a larger number of processors (up 1024), memory, and improved cache utilization, leading to a much better overall computational efficiency. Our basic strategy of parallelization is domain decomposition, similar to what was previously utilized by Homman *et al.* (5). The whole computational domain is divided into thin slabs in one direction and number of slabs corresponds to number of processors used. The differences between our implementation and Homman's lie in velocity interpolation and computation of particle pair statistics. Our new code computes in parallel radial distribution function (RDF (6) - a measure of the deviation of density distribution from uniform distribution), pair relative velocity, and dynamic collision rate. The parallel efficiency of these computations will be examined here.

2 Methodology

2.1 Flow Simulation

The usual pseudo-spectral method is employed to perform DNS of a forced isotropic homogenous turbulent flow. The incompressible Navier-Stokes and the continuity equation:

$$\frac{\partial \mathbf{U}}{\partial t} = \mathbf{U} \times \omega - \nabla \left(\frac{P}{\rho} + \frac{1}{2}\mathbf{U}^2 \right) + \nu \nabla^2 \mathbf{U} + \mathbf{f}\,(x,t)\,, \tag{1}$$

$$\nabla \cdot \mathbf{U} = 0 \tag{2}$$

are solved in a periodic cubic box. The flow domain is discretized uniformly into N^3 grid points, where in this study N takes the value of 128, 256, or 512. ω and P denote the fluid vorticity and pressure, respectively. The random forcing term $\mathbf{f}\,(x,t)$ is nonzero only for very low wave numbers (*i.e.*, $k \leq 1.5$), providing an energy source to sustain the air turbulence. Further details can

be found in (7). The main difference between the current and previous implementation is the implementation of FFT (fast fourier transform). FFT demands free access to data from every grid point in the whole computational domain. Domain decomposition limits access of a given process to data in a given subdomain. This difficulty was overcome by splitting the full 3D (three dimensional) FFT into a series of 2D FFTs, parallel matrix transposition and then 1D FFTs. The transposition step reorganizes data to facilitate 2D and 1D FFTs within each process. To minimize the transpose operation, the domain is decomposed into slabs in the k_z direction in the wavevector space, but along the $y-$direction in the physical space (8). The kinetic energy from low wave numbers propagates to small scales until viscous dissipation becomes active, eventually a quasi-steady kinetic energy balance is reached and flow becomes statistically stationary. This typically takes about 4 to 5 eddy turnover times after a random initialization of the flow field.

2.2 Particle Tracking

When the turbulent flow reaches the statistically stationary stage, we introduce particles at random positions with a uniform distribution. The random numbers for setting the initial particle location are generated only by one master process. Then, the master process sends position data for particles within a given subdomain to an appropriate process. Although, such a treatment is not parallel, it is usually fast and only needs to be performed once. For example, generating positions of 5 million particles and distributing them among 64 processors takes only 0.7 [s]. This method ensures a true random distribution in the whole computational domain as only one seed for the random number generator is needed.

Assuming that the particles are small in comparison with the Kolmogorov microscale and particle density is much larger than the fluid density, the following equation of motion (9)

$$\begin{cases} \frac{d\mathbf{V}(t)}{dt} = \frac{\mathbf{u}(\mathbf{Y},t)-\mathbf{V}(t)+\mathbf{W}}{\tau_p} \\ \frac{d\mathbf{Y}(t)}{dt} = \mathbf{V}(t) \end{cases} \tag{3}$$

is solved, where $\mathbf{V}(t)$ and $\mathbf{Y}(t)$ are the velocity and the centre position of particle, respectively, τ_p is the particle inertial response time, $\mathbf{u}$ is the fluid velocity at the particle location, $\mathbf{W} = \tau_p \mathbf{g}$ is the still-fluid terminal velocity, and $\mathbf{g}$ is gravitational acceleration. Equation 3 was solved by the 4th order Adams Bashford method. The initial particle velocity is set to the fluid velocity at the particle location plus the terminal velocity.

2.3 Velocity Interpolation

To compute the drag force acting on a particle, the fluid velocity at the particle location has to be interpolated from the solved fluid velocity field on a regular grid. Several different interpolation techniques have been developed previously, some of the more popular ones are shown in Table 1. In our MPI code

Table 1. Available interpolation schemes for fluid velocity at particle position

Yeung, P. K. and S. B. Pope 1989 (10)	Third order, thirteen-point fourth-order cubic spline
Balachandar S, Maxey MR, 1989 (11)	6-pt Lagrangian interpolation
Squires, K. D. and J. K. Eaton 1990 (12)	Tri-linear
Rovelstad et al. 1994 (13)	Spectral, Tri-linear, Cubic spline, Hermite - mathematically equivalent to tri-cubic
Rouson et al. 1997,2008 (14; 15)	Tri-linear
Bec J 2005 (16)	Spectral (few modes)
Franklin et al. 2005, 2007 (17; 18)	Tri-linear
Lekien and Marsden 2005 (19)	Tri-cubic (implementation no DNS)
Busse et al. 2007 (20)	Tri-cubic
Homann et al 2007a, 2007b (5; 21)	Tri-cubic an tri-linear

we implemented a Lagrangian interpolation scheme using 6 grid points in each spatial direction (7; 11). This requires data communication between neighboring processes in order to make sure that the full stencil of grid-based fluid velocity is always available when a particle is located in the vicinity of subdomain boundaries.

2.4 Periodic Boundary Condition

Particles moving in the turbulent flow field constantly change their subdomains or leave the computational box. When a particle moves into a different subdomain, the dynamic data occupied on the old process have to be transferred to a new process. Since the displacement of a particle during a time step is small, the particle will travel only to a neighboring slab. When a particle leaves the computational domain, periodic boundary condition is used to place the particle into a proper new process. Periodicity is realized by adding or subtracting the length of the computational box size from the particle coordinates. The communication time for moving data through processes depends on machine architecture and the number of nodes employed. Setting up separate communication between processors for sending and receiving data for every single particle is time consuming. Instead, we first made copies of the data in temporary buffer and then transferred the whole buffer in one operation. The original data table for particles within a given process is updated by removing selected items associated with particles that have left the subdomain, adding new particles just entered, and then re-indexing the whole table.

2.5 Collision Detection

The algorithm for collision detection implemented in the MPI code follows the idea presented by Wang et al. (22), with several modifications resulting from different memory management. Dynamic collision detection is executed in two steps. In the first step, collisions within a given subdomain are detected using an

efficient linked list method (23). In the second step the algorithm detects collisions occurring between particles from two different processes. The second step is faster because searching is limited only to a narrow slice covering overlapping region between two neighboring processors. To perform this step, the complete set of data with particles location, velocity and particles size has to be transferred to a neighboring process. The above treatment retains full parallelism and minimizes amount of data which has to be sent and received between CPUs.

2.6 Radial Distribution Function and Relative Velocity

Radial distribution function is computed based on the definition proposed in (6). Domain decomposition introduces additional boundaries inside the domain and complicates the detection of all particle pairs with separation distance in the range from $r-\delta$ to $r+\delta$, where δ is small fraction of collision radius $\sim 2\%$. This problem was solved again by a two-step procedure, similar to that used in the collision detection. Namely, particle pairs inside any given subdomain are detected first, then additional pairs involving particles from different processes are found. Relative velocity is computed for every pair used for the RDF calculation. These kinematic statistics are further averaged over time.

3 Parallel Performance

To examine the scalability of the MPI code, a number of numerical experiments were performed. All tests presented in this paper were conducted on an IBM Power 575 cluster (4064 POWER6 processors running at 4.7 GHz) at NCAR's supercomputing center.

In the first test we compare times designated for five major parts of the code: evolution of the turbulent flow, collision detection, velocity interpolation, advancing particle position and the implementation of the periodic boundary condition. The wall clock times using different number of processors and a given number of time steps are collected to determine the scalability for each of the tasks separately. The total measured time is the sum of computational and communication time. Additional time spent for saving data is negligible compared to either the computational or communication time. Maximal number of processors used in the test is 128 (4 full nodes). Figure 1 shows the total time needed for each task as a function of the number of processors.

The time spent for tasks related to the particle motion (collision detection, velocity interpolation, advancing particle position and periodic boundary condition) is inversely proportional to the number of processors. For the flow evolution the time also decreases but only for small number of processors (up to 32). For larger number of processors (64 and 128), the computational time appears to saturate, due to the increasing communication time associated with parallel FFT.

On the right panel of Fig. 1 we show the ratio of total computational time over the total execution (computational plus communication) time as a function of the number of processors. The computational time takes about half of the time for

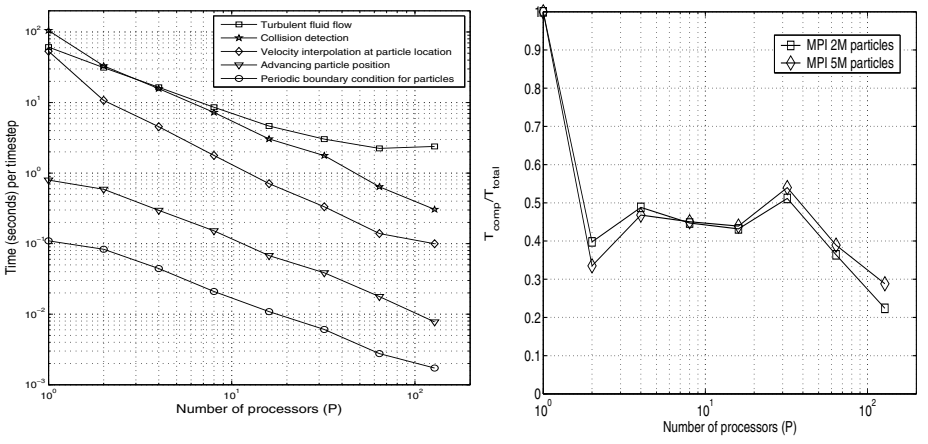


Fig. 1. Left panel: Scalability of the five major tasks in the DNS code in terms of the total execution time. Right panel: The ratio of $T_{computation}/T_{total}$ as a function of the number of processors. The grid size is 512^3.

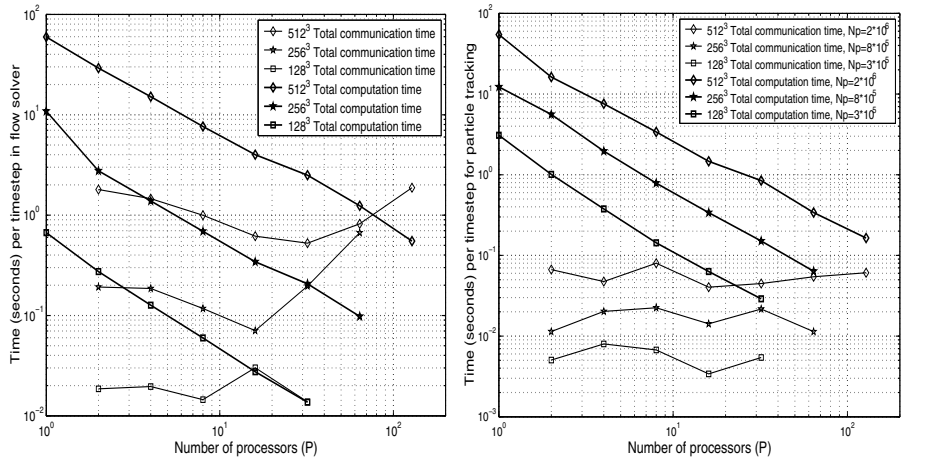


Fig. 2. Scalability of computational and comunication time for the flow solver (left panel) and the particle part (right panel), as a function of the number of processors

intermediate numbers of processors, but less than one third for larger numbers of processors. This shows that it is very critical to minimize the communication overhead in the MPI code.

In the second analysis, computational time and communication time for three different grid sizes 128^3, 256^3 and 512^3 are examined separately. Here the tasks are only divided into two groups: the flow solver part and particle part. Separate scaling performances are shown in Fig. 2. The computational time for both parts decreases roughly linearly with the number of processors, in the log-log plots. The dependence of the communication time on the number of processors

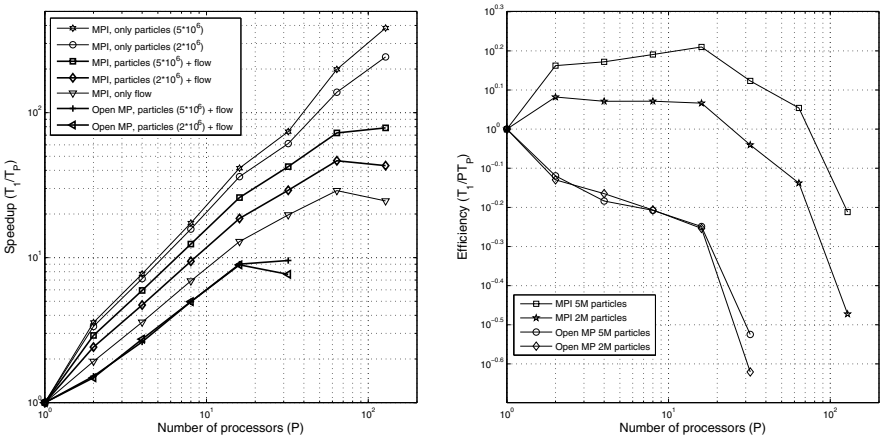


Fig. 3. Comparison of speedup (left panel) and computational efficiency (right panel) of the DNS codes parallelized with two different libraries (MPI and OpenMP). Grid is 512^3.

is more complex. For the flow evolution part, a reduction of communication time is realized with an increase in the number of processors from 1 to 32. This can be explained because amount of data being transferred between processes decreases with increasing number of processors. For larger number of processors more connections have to be established and the communication time in fact increases eventually with the number of processes. For the particle part the communication time is insensitive to the number of processors used.

In the third analysis, we compare the performance of two codes: our new MPI code and an OpenMP code developed previously by Ayala *et al.* (4). Both codes are functionally the same and yield identical results of collision statistics. Here we compare the speedup, efficiency (Fig. 3), and the memory usage (Fig. 4) of the codes, under an identical setting (the same initial flow field, the same number of particles, the same initial particle location and the same particle size). The speedup presented in Fig. 3 is determined by simulating 2 and 5 million particles starting from the same particle location and the same flow velocity field. Results obtained with OpenMP code show that, for the maximum allowed number of processors (32 on a single node), the speedup cannot exceed 10. In contrast, the MPI code can be run on more than one node and as such can be run on any number of processes. Figure 3 shows that, for the MPI code, the highest achievable speedup for 2 million particles occurs at 64 processors while for 5 million particles the highest speedup is for 128 processors. We can conclude that the speedup depends on the number of particles tracked. For 20 million particles, we expect that more than 128 processors can be used to achieve an efficiency close to 100%. In the Open MP code, efficiency decreases monotonically with the number of processes and it does not exceed 30%.

The small difference between two MPI curves in Fig. 4 shows that the majority of memory is used for flow solver rather than for the particle part. The memory

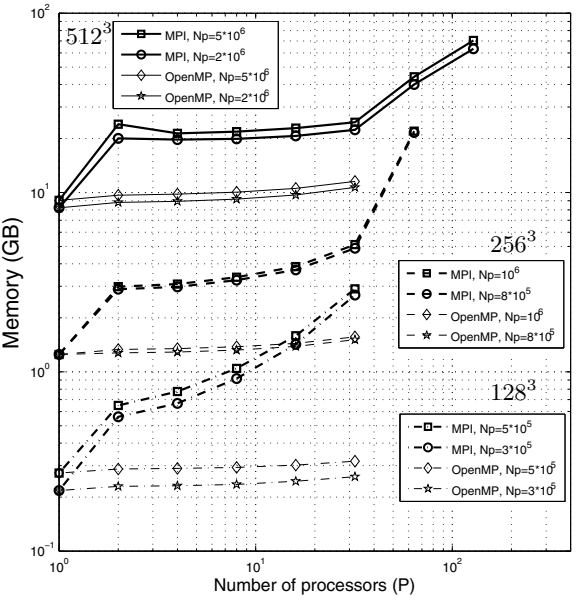


Fig. 4. Memory usage

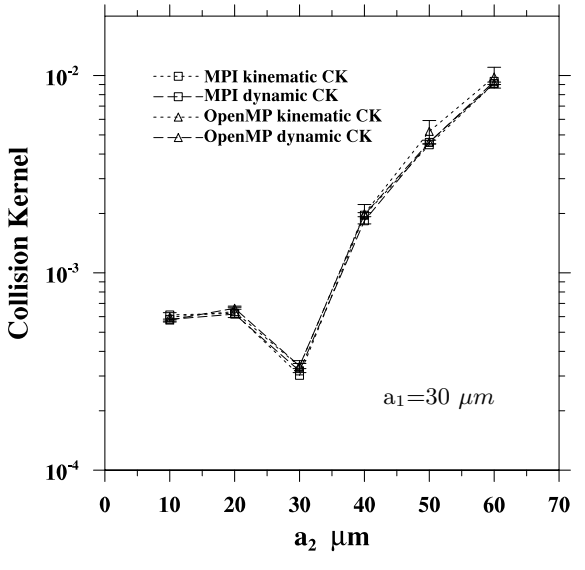


Fig. 5. Comparison of dynamic and kinematic collision kernels for sedimenting droplets in turbulent air computed in the MPI and OpenMP codes

usage for the particle part will not exceed 10 GB for 20 million particles. The high increase of memory usage between 32 processors and 128 is due to the allocation of additional buffers needed for communication. In the MPI code

the matrices with particle data was oversized by around 30 % in order to handle the particles which are moving between processes. Since in the Open MP code there is no need to allocate such oversized matrices and particle data size is defined precisely, memory usage is about 10 GB and is significantly less than in the MPI code.

Finally, we compare dynamic and kinematic collision kernels computed using the two codes (MPI and OpenMP). Figure 5 shows that the two codes give the same results and differences are within the statistical uncertainties of the data.

4 Conclusions

MPI implementation has been developed for a code designed to study turbulent collision of particles in a turbulent flow. This is a challenging task since turbulent flow and particle transport require different implementation strategies. Here we discussed major issues and implementation details, along with considerations for minimizing the communication overhead. A number of numerical tests are used to show scalability, speedup, and overall efficiency, and times required for different tasks are compared. The MPI code performs better than an earlier OpenMP code, and can take full advantage of distributed memory hybrid computers. Our next step is the MPI implementation for particle-particle aerodynamic interaction which requires considerations of both short and long-range interactions.

Acknowledgements

This work was supported by the National Science Foundation (NSF) under grants ATM-0527140 and NSF ATM-0730766. Computing resources are provided by National Center for Atmospheric Research (NCAR CISL-35751010).

References

[1] Ayala, O., Rosa, B., Wang, L.P., Grabowski, W.W.: Effects of turbulence on the geometric collision rate of sedimenting droplets. Part 1. Results from direct numerical simulation. New Journal of Physics 10 (2008)
[2] Ayala, O., Rosa, B., Wang, L.P.: Effects of turbulence on the geometric collision rate of sedimenting droplets. Part 2. Theory and parameterization. New Journal of Physics 10 (2008)
[3] Collins, L.R., Keswani, A.: Reynolds number scaling of particle clustering in turbulent aerosols. New Journal of Physics 6 (2004)
[4] Ayala, O., Grabowski, W.W., Wang, L.-P.: A hybrid approach for simulating turbulent collisions of hydrodynamically-interacting particles. JCP, 51–73 (2007)
[5] Homann, H., Dreher, J., Grauer, R.: Impact of the floating-point precision and interpolation scheme on the results of DNS of turbulence by pseudo-spectral codes. Computer Physics Communications 177(7), 560–565 (2007)
[6] Wang, L.-P., Wexler, A.S., Yong, Z.: Statistical mechanical description and modelling of turbulent collision of inertial particles. J. Fluid Mech., 117–153 (2000)

[7] Wang, L.P., Maxey, M.R.: Settling velocity and concentration distribution of heavy-particles in homogeneous isotropic turbulence. Journal of Fluid Mechanics 256, 27–68 (1993)
[8] Dmitruk, P., Wang, L.P., Matthaeus, W.H., Zhang, R., Seckel, D.: Scalable parallel FFT for spectral simulations on a Beowulf cluster. Parallel Computing 27(14), 1921–1936 (2001)
[9] Maxey, M.R., Riley, J.J.: Equation of motion for a small rigid sphere in a nonuniform flow. Physics of Fluids 26(4), 883–889 (1983)
[10] Yeung, P.K., Pope, S.B.: Lagrangian statistics from direct numerical simulations of isotropic turbulence. Journal of Fluid Mechanics 207, 531–586 (1989)
[11] Balachandar, S., Maxey, M.R.: Methods for evaluating fluid velocities in spectral simulations of turbulence. JCP, 96–125 (1989)
[12] Squires, K.D., Eaton, J.K.: Particle response and turbulence modification in isotropic turbulence. Physics of Fluids a-Fluid Dynamics 2(7), 1191–1203 (1990)
[13] Rovelstad, A.L., Handler, R.A., Bernard, P.S.: The effect of interpolation errors on the lagrangian analysis of simulated turbulent channel flow. Journal of Computational Physics 110(1), 190–195 (1994)
[14] Rouson, D.W.I., Kassinos, S.C., Moulitsas, I., Sarris, I.E., Xu, X.: Dispersed-phase structural anisotropy in homogeneous magnetohydrodynamic turbulence at low magnetic Reynolds number. Physics of Fluids 20(2), 19 (2008)
[15] Damian, W.I., Rouson, J.K.E., Abrahamson, S.D.: A direct numerical simulation of a particle-laden turbulent channel flow. No. TSD-101 (1997)
[16] Bec, J.: Multifractal concentrations of inertial particles in smooth random flows. Journal of Fluid Mechanics 528, 255–277 (2005)
[17] Franklin, C.N., Vaillancourt, P.A., Yau, M.K., Bartello, P.: Collision rates of cloud droplets in turbulent flow. Journal of the Atmospheric Sciences 62(7), 2451–2466 (2005)
[18] Franklin, C.N., Vaillancourt, P.A., Yau, M.K.: Statistics and parameterizations of the effect of turbulence on the geometric collision kernel of cloud droplets. Journal of the Atmospheric Sciences 64(3), 938–954 (2007)
[19] Lekien, F., Marsden, J.: Tricubic interpolation in three dimensions. International Journal for Numerical Methods in Engineering 63(3), 455–471 (2005)
[20] Busse, A., Muller, W.C., Homann, H., Grauer, R.: Statistics of passive tracers in three-dimensional magnetohydrodynamic turbulence. Physics of Plasmas 14(12) (2007)
[21] Homann, H., Grauer, R., Busse, A., Mueller, W.: Lagrangian statistics of Navier-Stokes and MHD turbulence. Journal of Plasma Physics 73, 821–830 (2007)
[22] Wang, L.-P., Franklin, C.N., Ayala, O., Grabowski, W.W.: Probability distributions of angle of approach and relative velocity for colliding droplets in a turbulent flow. JAS, 881–900 (2006)
[23] Allen, M.P., Tildesley, D.J.: Computer simulation of liquids, p. 408. Oxford University Press, New York (1987)

A Distributed Multilevel Ant-Colony Approach for Finite Element Mesh Decomposition

Katerina Taškova, Peter Korošec, and Jurij Šilc

Jožef Stefan Institute, Computer Systems Department
Jamova cesta 39, SI-1000 Ljubljana, Slovenia
{katerina.taskova,peter.korosec,jurij.silc}@ijs.si

Abstract. The k-way finite element mesh (FEM) decomposition problem is an NP-complete problem, which consists of finding a decomposition of a FEM into k balanced submeshes such that the number of cut edges is minimized. The multilevel ant-colony algorithm (MACA) is quite new and promising hybrid approach for solving different type of FEM-decomposition problems. The MACA is a swarm-based algorithm and therefore inherently suitable for parallel processing on many levels. Motivated by the good performance of the MACA and the possibility to improve it's performance (computational cost and/or solution quality), in this paper we discuss the results of parallelizing the MACA on largest scale (on colony level). Explicitly, we present the distributed MACA (DMACA) approach, which is based on the idea of parallel independent runs enhanced with cooperation in form of a solution exchange among the concurrent searches. Experimental evaluation of the DMACA on a larger set of benchmark FEM-decomposition problems shows that the DMACA compared to the MACA can obtain solutions of equal quality in less computational time.

1 Introduction

Finite element method is a well known numerical method that efficiently solves complex system of partial differential equations. Based on the problem, generated mesh structure can have large number of elements, making them computationally expensive. Therefore a common approach involves decomposition of a large-scale finite element mesh (FEM) into less complex and well balanced submeshes that can be solved in a multiprocessor environment with minimal possible inter-processor communication. Unfortunately, this makes the FEM-decomposition problem a NP-hard combinatorial optimization problem which is computationally expensive, as well.

A variety of metaheuristic methods are used for solving the FEM-decomposition problem [1,5,6,7]. Recently, based on the ant-colony optimization empowered with the multilevel approach for faster convergence and problem size reduction, the multilevel ant-colony algorithm (MACA) [8] was proposed for solving FEM-decomposition problem.

The MACA is a swarm-based algorithm and therefore inherently suitable for parallel processing on many levels. Motivated by the good performance of the

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 398–407, 2010.

MACA in the previous work [8] and the possibility to improve it's performance (computational cost and/or solution quality), we discuss the results of paralleliz-ing the MACA on largest scale (on colony level [10]), executing entire algorithm runs concurrently on a cluster of workstations. Explicitly, we present and experimentally evaluate the *distributed MACA* (DMACA), initially introduced in [12] as the semi-independent DMACA.

We have to stress out that the purpose of this study was not to compare the DMACA with state-of-the-art optimization methods used for FEM decomposition, but to explore the possibility of improvement of the MACA method exploiting the inherent property of parallelism given a distributed memory multiprocessor environment. On the other hand adequate experimental evaluation and comparison of the MACA algorithm was performed in [8], where was shown that the MACA is superior to the classical k-METIS and Chaco 2.0 methods, and comparable to the MLSATS algorithm and to the JOSTLE Evolutionary algorithm. (References for the methods can be find in [8].)

The rest of the paper is organized as follows. Section 2 describes in brief the MACA algorithm and Section 3 introduces the DMACA algorithm for solving the FEM-decomposition problem. The experimental setup along with the obtained results are presented and discussed in Section 4. Conclusions and possible directions for further work are given in Section 5.

2 Sequential Multilevel Ant-Colony Algorithm

The main idea of the ant-colony algorithm for k-way FEM decomposition is very simple [9]. We have k colonies of ants that perform probabilistic moves on a grid (represents the ants' habitat) and compete for food (initially randomly placed on the grid cells), which in this case is represented by the elements of the mesh. Final outcome of ants activities is stored food in their nests, i.e., they decompose the mesh into k submeshes.

```
 1: structure[0] = Initialization()
 2: for ℓ = 0 to L − 1 do
 3:    structure[ℓ + 1] = Coarsening(structure[ℓ])
 4: end for
 5: for ℓ = L downto 0 do
 6:    Solver(structure[ℓ]) //basic ACO method
 7:    if ℓ > 0 then
 8:       structure[ℓ − 1] = Refinement(structure[ℓ])
 9:       BucketInitialization()
10:    end if
11: end for
```

Algorithm 1. MACA

The MACA algorithm is an ant-colony algorithm for k-way FEM decomposition enhanced with a multilevel technique [15] to promote faster convergence of the optimization metaheuristic and solution to a larger problems. It is

a recursive-like procedure that combines four basic methods: graph partitioning (*solver*, i.e., the basic ACO method), graph contraction (*coarsening*), graph expansion (*refinement*) and vertex arrangement (*bucket sorting*). Algorithm 1 outlines top-level MACA pseudo code. Details on the particular methods can be found in [8].

3 Distributed Multilevel Ant-Colony Algorithm

Initial study on parallelization of the MACA [12] examined two distributed versions of the MACA.

Master:

```
1: StartAllSlaves()
2: while not ending condition do
3:    while all slaves not finished level do
4:       ReceiveFromSlave(BestLevelSlaveMeshDecomposition)
5:       AllBestLevelMeshDecomposition=Add(BestLevelSlaveMeshDecomposition)
6:    end while
7:    LevelBestMeshDecomposition=Calculate(AllBestLevelMeshDecomposition)
8:    BroadcastToSlaves(LevelBestMeshDecomposition)
9:    if last level finished then
10:      BestMeshDecomposition=LevelBestMeshDecomposition
11:   end if
12: end while
13: StopAllSlaves()
```

Slave:

```
1: ReceiveFromMaster(Parameters)
2: MeshStructure[0]=Initiliazation (Parameters)
3: for ℓ = 0 to L − 1 do
4:    MeshStructure[ℓ + 1] = Coarsening(MeshStructure[ℓ])
5: end for
6: for ℓ = L downto 0 do
7:    BestLevelMeshDecomposition=Solver(MeshStructure[ℓ]) //basic ACO method
8:    SendToMaster(BestLevelMeshDecomposition)
9:    ReceiveFromMaster(BestLevelMeshDecomposition)
10:   Update(MeshStructure[ℓ],BestLevelMeshDecomposition)
11:   if ℓ > 0 then
12:      MeshStructure[ℓ − 1] = Refinement(MeshStructure[ℓ])
13:      BucketInitialization()
14:   end if
15: end for
```

Algorithm 2. DMACA

The first one was based on the parallel interactive colony approach which, by definition, implied master/slave implementation and synchronized communication. Disadvantage of this version was the synchronization/communication overhead, since information exchange across the concurrent processors was initiated every time a piece of food had been taken or dropped on a new position.

Furthermore, the master kept and updated its own local grid matrix of temporal food positions (played a role of a shared memory) in order to maintain normal and consistent slaves activities.

Trying to avoid the communication and still exploit some level of parallelism, the second version distributes the MACA on the idea of parallel independent runs [11] enhanced with cooperation in form of a solution exchange among the concurrent searches. In this paper we consider the second approach, referred to as the DMACA.

The DMACA is basically approach that allows exchange of the best temporal solution at the end of every level of the multilevel optimization process. This exchange requires that the parallel executions of the MACA instances on the available processors have to be synchronized once per level. Namely, the master processor is responsible for synchronizing the work of all slave processor that execute a copy of the DMACA, by managing the exchange information and communication process, while the slave processors have to execute the instances of the DMACA code, signal when finish the current level optimization and send the best partition to the master. When all finish the current level, the master determines the best solution and broadcasts it to the slaves. In order to proceed with next level optimization, slave processors have to first update local memory structures (grid matrix) and afterwards perform partition expansion (refinement). The main idea of the DMACA is outlined in Algorithm 2.

4 Experimental Evaluation

Proposed DMACA was applied on a set of benchmark FEM-decomposition problems and the results from experimental evaluation on 2-way and 4-way FEM decomposition are presented and discussed in this section.

4.1 Performance Measures

The solution *quality* is evaluated with regard to the FEM decomposition quality measures: the total number of edges connecting the submeshes (cut-size) and the maximal difference in the size of the obtained submeshes (imbalance). Since the imbalance of the final obtained solutions is kept in a predefined range of values, we report the quality in terms of the cut-size.

Statistical significance test was performed to check the difference in the quality of the obtained solutions by the MACA and the DMACA. We used pairwise comparisons with the Wilcoxon signed-ranks test [16] and multiple comparisons with the Bergmann-Hommel dynamic post-hoc procedure [3].

Reporting results from experiments with parallel algorithms is not a straightforward task [2]. Moreover, in the case of stochastic algorithms, as the DMACA is, the repeatability of its outcome is questionable, making the performance evaluation procedure even more difficult. The standard way of reporting results with the mean value and the corresponding variance of the best found solutions over all performed executions (runs) is not always sufficient (the mean value could be

far away from the global optimal). Therefore, instead of maximizing the number of function evaluations that guarantee a success (finding the global best solution, f_g), the aim is to find the number of runs that maximize the success rate (probability of hitting the global optimum). For a given tolerance tol_f, the success rate is calculated as the fraction of m runs that found solution in the range between $-\infty$ and $f_g + tol_f$, where tol_f is problem dependent.

Finally, the effectiveness of the parallel algorithm is in our case given by the *speed-up* measure

$$S(n) = \frac{t_{\mathrm{S}}}{t_{\mathrm{T}}(n)}$$

and by the *relative speed-up* measure

$$S_r(n) = \frac{t_{\mathrm{T}}(1)}{t_{\mathrm{T}}(n)}.$$

Here, t_{S} is the time to solve a problem with the sequential code (MACA), $t_{\mathrm{T}}(1)$ is time to solve a problem with the parallel code (DMACA) with one processor, and $t_{\mathrm{T}}(n)$ is time to solve the same problem with the parallel code on n processors. According to the study [2], the speed-up results were calculated based on the mean value of the time for the serial code, while the final result was presented as harmonic mean of the speed-up values of all runs.

4.2 Setup

Based on the MACA sequential code, the DMACA is implemented in Borland Delphi, using TCP/IP protocol for the server/client communication, based on the open source library Indy Sockets 10. All experiments were performed on a 8-node cluster connected via a Giga-bit switch, where each node consists of two AMD Opteron 1.8-GHz processors, 2GB of RAM, and Windows XP operating system.

The benchmark FEMs used in the experimental analysis were taken from [4,13] (Table 1).

Table 1. Description of the benchmark FEMs

FEM	Description	Vertices	Edges
grid1	Grid graph	252	476
grid2	Grid graph	3296	6432
U1000.05	Random geometric graph	1000	2394
U1000.10	Random geometric graph	1000	4696
U1000.20	Random geometric graph	1000	9339
crack	2D nodal graph	10240	30380
data		2851	15093
vibrobox	Sparse matrix	12328	165250
wing_nodal	3D nodal graph	10937	75488
bcsstk33	3D stiffness matrix	8738	291583
crack_dual	2D nodal graph	20141	30043
big	Numerical grid	15606	45878

The total number of ants per colony was 120. The number of ants per subcolony was determined from the number n of processors as $\frac{1}{n}$ of the total number of ants. We set a fixed number (600) of iterations per colony per ant per level in both, the MACA and the DMACA, except on the last level when the algorithms were driven by the solution quality improvement and stopped when in the last successive 600 iterations no improvement was obtained. Additionally, the search of the optimal submesh was limited inside of 1.5% imbalance only on the last level. The experiments were applied on all FEMs and executed 200 times in order to guarantee an error of 0.05 on the probability of success with 95% level of confidence [14].

4.3 Results

Results on cut-size performance for the MACA and the DMACA are presented in Table 2 and Table 3, respectively.

Table 2. Cut-size obtained by the MACA for 2-way and 4-way FEM decomposition and 1.5% imbalance. The best, mean and standard deviation values are calculated from 200 runs.

	$k = 2$			$k = 4$		
FEM	best	mean	std	best	mean	std
grid1	18	18.01	0.07	38	39.75	1.27
grid2	34	45.97	13.42	93	104.64	6.55
U1000.05	1	1.30	0.51	7	14.47	5.17
U1000.10	39	61.98	7.54	80	117.23	14.31
U1000.20	219	268.66	37.81	463	580.30	83.76
crack	184	204.28	19.36	374	433.86	45.32
data	212	247.37	7.08	406	489.98	42.25
vibrobox	11249	11963.54	254.88	20061	21086.26	375.40
wing_nodal	1712	1753.23	26.93	3622	3817.25	141.18
bcsstk33	10883	11299.23	635.51	22741	24581.27	860.46
crack_dual	80	90.80	8.37	171	205.49	22.27
big	139	254.28	41.40	341	444.42	45.88

Based on cut-size results, Tables 4 and 5 present pairwise comparisons with the Wilcoxon signed-ranks test and multiple comparisons with the Bergmann-Hommel dynamic post-hoc procedure, respectively. The test confirms that in general there is not significant difference in the quality of generated solutions with the MACA and the DMACA. For 1% significance level all hypotheses are retained.

When compared the MACA and the DMACA with respect to the success rate (Table 6) is evident that fair discrimination between them is highly associated with the determination of the tol_f factor. In our case we set $tol_f = 0.01$ and

Table 3. Cut-size obtained by the DMACA for 2-way and 4-way FEM decomposition and 1.5% imbalance. The best, mean and st. deviation values are calculated from 200 runs.

		$n=1$			$n=2$			$n=4$			$n=8$			$n=16$		
k	FEM	best	mean	std	best	mean	std	best	mean	std	best	mean	std	best	mean	std
	grid1	18	18.00	0.00	18	18.00	0.00	18	18.01	0.07	18	18.09	0.30	18	18.91	1.18
	grid2	34	42.16	11.74	34	37.41	6.24	35	38.15	6.42	35	51.42	13.45	37	65.79	8.33
	U1000.05	1	1.21	0.42	1	1.04	0.20	1	1.00	0.00	1	1.00	0.00	1	1.00	0.00
	U1000.10	39	61.17	8.10	39	57.18	8.05	39	59.53	6.96	39	58.64	5.41	40	57.35	4.34
	U1000.20	219	265.87	42.14	219	277.09	53.69	219	273.63	52.96	218	269.26	53.28	219	268.59	50.96
2	crack	184	194.04	16.34	184	195.92	11.85	184	199.22	12.27	185	204.32	13.02	188	216.37	12.68
	data	212	247.43	7.71	204	246.42	14.06	212	234.51	21.28	216	222.48	9.41	216	222.76	6.00
	vibrobox	11382	11900.42	202.27	11258	11858.32	164.75	11684	11967.59	170.19	11834	12214.93	170.06	12011	12496.04	178.59
	wing_nodal	1714	1756.33	28.43	1714	1773.73	41.79	1719	1785.46	42.75	1734	1787.06	38.33	1754	1795.69	36.41
	bcsstk33	10873	11235.24	674.62	10565	11328.95	750.33	10594	11232.47	791.38	10610	11046.40	562.88	10850	11068.33	372.52
	crack_dual	80	90.04	8.07	79	88.93	4.77	80	90.76	5.17	80	90.88	4.03	82	92.07	7.53
	big	139	257.11	37.11	140	241.15	42.09	140	241.81	37.35	143	240.53	36.50	144	247.14	35.99
	grid1	38	39.47	1.12	38	38.33	0.63	38	38.09	0.30	38	38.05	0.30	38	38.12	0.38
	grid2	92	104.97	6.59	92	104.82	6.18	92	103.77	6.73	92	104.45	7.73	93	107.14	7.81
	U1000.05	7	14.37	5.31	7	14.47	4.91	7	13.49	3.74	6	12.23	2.82	7	11.65	2.27
	U1000.10	91	115.91	12.26	88	112.15	11.71	87	111.03	11.87	87	111.80	10.96	84	111.19	10.99
	U1000.20	485	596.83	97.31	481	591.97	80.70	481	615.54	78.42	480	628.09	85.84	479	620.09	92.97
4	crack	371	418.74	37.97	372	413.52	32.48	371	415.19	28.73	373	422.44	32.15	375	431.31	35.02
	data	407	481.61	37.38	407	468.58	36.73	408	461.57	33.46	407	467.57	32.52	409	469.73	31.26
	vibrobox	20053	20953.22	407.35	19942	20919.01	381.28	19938	20998.80	373.44	19988	21208.94	325.10	20593	21383.95	270.49
	wing_nodal	3647	3774.56	115.49	3672	3745.40	43.06	3666	3748.17	29.48	3693	3758.59	29.82	3707	3768.20	29.17
	bcsstk33	22313	24346.88	697.50	23217	24105.18	470.44	23130	23901.77	464.97	23011	23954.20	537.56	23163	23963.14	553.64
	crack_dual	171	200.06	19.21	168	193.55	17.50	170	196.15	14.95	177	202.13	15.31	178	210.38	19.43
	big	347	443.23	40.56	342	450.28	40.91	344	442.85	39.38	347	441.18	45.13	347	444.45	42.05

Table 4. Pairwise comparisons with the Wilcoxon test

	$k=2$			$k=4$		
Hypothesis	R^+	R^-	p-value	R^+	R^-	p-value
MACA vs. DMACA$_{n=1}$	16	62	0.077	12	66	0.034
MACA vs. DMACA$_{n=2}$	28	50	0.424	10	68	0.042
MACA vs. DMACA$_{n=4}$	19	59	0.240	9	69	0.016
MACA vs. DMACA$_{n=8}$	39	39	1	20	58	0.151
MACA vs. DMACA$_{n=16}$	34	44	0.733	30	48	0.519

Table 5. Multiple comparisons with the Bergmann-Hommel procedure

	Adjusted p-value	
Hypothesis	$k=2$	$k=4$
DMACA$_{n=1}$ vs. DMACA$_{n=2}$	1	0.560
DMACA$_{n=1}$ vs. DMACA$_{n=4}$	1	0.030
DMACA$_{n=1}$ vs. DMACA$_{n=8}$	1	0.560
DMACA$_{n=1}$ vs. DMACA$_{n=16}$	1	1
DMACA$_{n=2}$ vs. DMACA$_{n=4}$	1	0.787
DMACA$_{n=2}$ vs. DMACA$_{n=8}$	1	1
DMACA$_{n=2}$ vs. DMACA$_{n=16}$	0.612	0.560
DMACA$_{n=4}$ vs. DMACA$_{n=8}$	1	0.787
DMACA$_{n=4}$ vs. DMACA$_{n=16}$	1	0.030
DMACA$_{n=8}$ vs. DMACA$_{n=16}$	1	0.560

global best solutions f_g were adopted according to the best available solutions for the considered FEM-decomposition problems. The results show that in case on 2-way decomposition of grid1, the MACA has better success rate than the DMACA$_{n=16}$, slightly better success rate than the DMACA$_{n=8}$ and equal good success rate with the DMACA$_{n=2,4}$. Similar in case on 2-way decomposition of crack, the MACA has better success rate than the DMACA$_{n=4,8,16}$ and in case of 4-way decomposition on U1000.20, the MACA has better success rate than the DMACA$_{n=2,4,8,16}$. The DMACA has better convergence than the MACA, in case of 2-way decomposition on U1000.05, 4-way decomposition on grid1, 4-way decomposition on grid2 ($n = 4, 8$), 4-way decomposition on crack ($n = 2, 4$), 4-way decomposition on data ($n = 2, 4, 8$). In the rest of the cases it is not possible to make a difference because, either the tolerance factor is too small or the threshold for the error on the success rate is to high. Having in mind that the quality of the DMACA is preserved, then any speed-up we can gain is a benefit. Results on speedup are presented in Table 7. We observe speedup in case when the DMACA executed on 4 or more processors. General observation is that parallel performance of the system with respect to speed-up over the serial MACA is poor compared to the theoretical expected speed-up. This is to some level expected, since MACA was optimally designed for single processor execution.

Table 6. Success rate obtained for 2-way and 4-way FEM decomposition and 1.5% imbalance, based on tol_f=0.01 and 200 runs. Legend – **bold**-value: DMACA better than or equal to MACA, *italic*-value: DMACA worse than MACA, normal-value: no decision

	$k=2$					$k=4$				
		DMACA					DMACA			
FEM	MACA	$n=2$	$n=4$	$n=8$	$n=16$	MACA	$n=2$	$n=4$	$n=8$	$n=16$
grid1	0.995	**1.000**	**0.995**	*0.915*	*0.495*	0.145	**0.745**	**0.915**	**0.965**	**0.905**
grid2	0.010	**0.010**	*0.000*	*0.000*	*0.000*	0.040	**0.060**	**0.110**	**0.105**	*0.025*
U1000.05	0.730	**0.960**	**1.000**	**1.000**	**1.000**	0.000	0.000	0.000	0.005	0.000
U1000.10	0.015	**0.035**	**0.020**	**0.010**	*0.000*	0.035	**0.055**	**0.080**	**0.030**	**0.065**
U1000.20	0.035	**0.040**	**0.070**	**0.075**	**0.055**	0.465	*0.395*	*0.270*	*0.250*	*0.315*
crack	0.225	**0.230**	*0.090*	*0.010*	*0.000*	0.030	**0.100**	**0.085**	**0.045**	*0.015*
data	0.000	0.000	0.000	0.000	0.000	0.045	**0.145**	**0.190**	**0.155**	**0.090**
vibrobox	0.025	*0.010*	*0.000*	*0.000*	*0.000*	0.005	**0.045**	**0.030**	**0.005**	**0.000**
wing_nodal	0.040	**0.065**	*0.020*	*0.000*	*0.000*	0.005	0.000	0.000	0.000	0.000
bcsstk33	0.000	0.000	0.000	0.000	0.000	0.005	0.000	0.000	0.000	0.000
crack_dual	0.005	**0.035**	**0.010**	**0.005**	**0.000**	**0.000**	**0.020**	**0.005**	**0.000**	**0.000**
big	0.045	*0.010*	*0.010*	*0.000*	*0.000*	0.035	*0.015*	**0.040**	**0.060**	*0.020*

Table 7. Speed-up obtained for 2-way and 4-way FEM decomposition and 1.5% imbalance, based on 200 runs

	$S_r(n)$								$S(n)$									
	$k=2$				$k=4$				$k=2$					$k=4$				
FEM $n=$	2	4	8	16	2	4	8	16	1	2	4	8	16	1	2	4	8	16
grid1	0.48	0.92	2.83	2.90	0.47	0.90	3.04	2.71	1.01	0.49	0.93	2.87	2.93	0.98	0.46	0.88	2.98	2.66
grid2	0.78	1.47	2.41	3.20	0.73	1.34	2.61	3.35	0.89	0.69	1.30	2.14	2.83	0.76	0.56	1.02	1.98	2.54
U1000.05	1.03	1.42	2.69	4.71	0.86	1.24	1.82	2.03	0.94	0.97	1.34	2.53	4.44	0.85	0.73	1.05	1.55	1.73
U1000.10	0.73	1.29	2.27	2.74	0.57	1.08	2.96	3.10	0.95	0.69	1.23	2.17	2.61	0.86	0.49	0.93	2.54	2.66
U1000.20	0.64	1.23	2.59	2.74	0.55	1.09	3.15	3.01	0.96	0.62	1.19	2.50	2.65	0.92	0.51	1.00	2.89	2.76
crack	1.03	1.64	2.38	3.50	1.11	1.80	2.65	4.24	0.79	0.81	1.30	1.88	2.76	0.81	0.90	1.46	2.15	3.44
data	0.66	1.22	1.91	2.53	0.67	1.26	2.41	3.25	1.01	0.66	1.23	1.92	2.55	0.76	0.51	0.96	1.83	2.47
vibrobox	1.30	1.62	1.86	2.16	1.23	1.67	2.06	2.69	0.81	1.06	1.32	1.51	1.76	0.75	0.93	1.26	1.55	2.02
wing_nodal	1.08	1.54	2.14	2.81	1.22	1.81	2.65	3.77	0.84	0.91	1.29	1.80	2.37	0.72	0.87	1.30	1.90	2.71
bcsstk33	1.00	1.19	1.29	1.39	0.95	1.16	1.30	1.42	0.86	0.86	1.02	1.11	1.19	0.89	0.85	1.04	1.16	1.27
crack_dual	0.99	1.79	2.45	3.25	1.16	2.13	3.08	4.52	0.77	0.77	1.39	1.90	2.52	0.76	0.88	1.62	2.35	3.45
big	1.03	1.85	2.36	3.09	1.12	1.86	2.77	4.14	0.79	0.82	1.47	1.87	2.46	0.74	0.83	1.38	2.05	3.06

5 Conclusions

This paper presents the distributed multilevel ant colony algorithm (DMACA) which is based on the idea of parallel independent runs enhanced with cooperation in form of a solution exchange among the concurrent searches. The experimental evaluation shows that the DMACA can get the quality obtained by sequential algorithm, while decreasing the overall computational cost.

Since distributed implementation suffers from increased communication and local memory updates (as evident in [12]), logical and possible further step will be to test a corresponded shared memory implementation.

References

1. Bahreininejad, A., Topping, B.H.V., Khan, A.I.: Finite Element Mesh Partitioning Using Neural Networks. Adv. Eng. Softw. 27, 103–115 (1996)
2. Barr, R., Hickman, B.: Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinion. ORSA J. Comput. 5, 2–18 (1993)
3. Bergmann, B., Hommel, G.: Improvements of General Multiple Test Procedures for Redundant Systems of Hypotheses. In: Bauer, P., Hommel, G., Sonnemann, E. (eds.) Multiple Hypothesenprfung–Multiple Hypotheses Testing, pp. 100–115. Springer, Berlin (1988)
4. Graph Partitioning – Graph Collection. Visited (October 2009), http://wwwcs.uni-paderborn.de/cs/ag-monien/RESEARCH/PART/graphs.html
5. Kadłuczka, P., Wala, K.: Tabu Search and Genetic Algorithms for the Generalized Graph Partitioning Problem. Control Cybern. 24, 459–476 (1995)
6. Kaveh, A., Shojaee, S.: Optimal Domain Decomposition via p-median Methodology Using ACO and Hybrid ACGA. Finite Anal. Elem. Des. 44, 505–512 (2008)
7. Kernighan, B.W., Lin, S.: An Efficient Heuristic Procedure for Partitioning Graph. Bell Sys. Tech. J. 49, 291–307 (1970)
8. Korošec, P., Šilc, J., Robič, B.: Solving the Mesh-partitioning Problem with an Ant-colony Algorithm. Parallel Comput. 30, 785–801 (2004)
9. Langham, A.E., Grant, P.W.: Using Competing Ant Colonies to Solve k-way Partitioning Problems with Foraging and Raiding Strategies. In: Floreano, D., Mondada, F. (eds.) ECAL 1999. LNCS, vol. 1674, pp. 621–625. Springer, Heidelberg (1999)
10. Randall, M., Lewis, A.: A Parallel Implementation of Ant Colony Optimization. J. Parallel Distr. Com. 62, 1421–1432 (2002)
11. Stützle, T.: Parallelization Strategies for Ant Colony Optimization. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 722–741. Springer, Heidelberg (1998)
12. Taškova, K., Korošec, P., Šilc, J.: A Distributed Multilevel Ant Colonies Approach. Informatica 32, 307–317 (2008)
13. The Graph Partitioning Archive (visited October 2009), http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/
14. Vasile, M., Minisci, E., Locatelli, M.: Testing Approaches for Global Optimization of Space Trajectories. In: Filipič, B., Šilc, J. (eds.) Bioinspired Optimization Methods and their Applications, pp. 81–91. Jožef Stefan Institute, Ljubljana (2008)
15. Walshaw, C., Cross, M.: Mesh Partitioning: A Multilevel Balancing and Refinement Algorithm. SIAM J. Sci. Comput. 22, 63–80 (2001)
16. Wilcoxon, F.: Individual Comparisons by Ranking Methods. Biometrics Bull. 1, 80–83 (1945)

Toward Definition of Systematic Criteria for the Comparison of Verified Solvers for Initial Value Problems

Ekaterina Auer[1] and Andreas Rauh[2]

[1] Faculty of Engineering, INKO, University of Duisburg-Essen,
D-47048 Duisburg, Germany
auer@inf.uni-due.de
[2] Chair of Mechatronics, University of Rostock,
D-18059 Rostock, Germany
andreas.rauh@uni-rostock.de

Abstract. Solving initial value problems for ordinary differential equations is a common task in many disciplines. Over the last decades, several different verified techniques have been developed to compute enclosures of the exact result numerically. The obtained bounds are guaranteed to contain the corresponding solution to the initial value problem. Ideally, we want to calculate *tight* enclosures over sufficiently *long* time intervals for systems with uncertainties in both the initial conditions and system parameters. However, the existing solvers are not always equal in attaining this goal. On the one hand, the quality of the obtained results depends strongly on the types of ordinary differential equations that describe a given dynamical system. On the other hand, a great influence of the considered uncertainties can be observed. Our general aim is to provide assistance in choosing an appropriate verified initial value problem solver with its most suitable 'tuning parameters' for the application at hand. In this paper, we make first steps toward setting up a framework for the fair comparison of the different approaches. We suggest criteria, benchmark scenarios, and typical applications which can be used for the quantification of the efficiency of verified initial value problem solvers.

1 Introduction

In many research projects and practical applications, verified solvers of initial value problems (IVPs) for ordinary differential equations (ODEs) have been employed efficiently to characterize the dynamic behavior of a given system with bounded uncertainties in initial conditions or other parameters as described, for instance, in [1]. However, these successful examples remain largely unnoticed by a typical industry engineer. One of the reasons for this is the lack of information on which tool to choose for a specific application. Especially, the choice of the 'optimal' settings for the solver, that is, conditions under which the best result (in a given sense) can be obtained, might prove to be difficult for a person unacquainted with the algorithm.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 408–417, 2010.

This is not a problem characterizing verified algorithms in particular. For floating point based tools, comparison criteria were being developed since the 1970s to allow users to choose the right program in dependence on their task. In the field of IVP solvers, the works on DETEST [2],[3] stand out. They were continued at the INdAM Bari by the project group *Codes and Test Problems for Differential Equations*, which resulted in the appearance of TESTSET [4].

In these works, the authors develop a set of example problems and a set of criteria to highlight the typical properties of the considered solvers in a unified manner. In DETEST and StiffDETEST, all systems are divided into stiff and non-stiff. Further, the non-stiff problems are classified into single equations, small systems, moderate systems, orbit equations and higher order equations, whereas the classes for stiff problems are linear with real/non-real eigenvalues, nonlinear coupling as well as nonlinear with real/non-real eigenvalues. The groups in TESTSET are defined according to the type of the considered IVP: for ordinary differential and explicit/implicit differential-algebraic equations.

The comparison criteria in DETEST are designed to describe the long-term behavior of a given solver. They are based on the number of function evaluations and the overhead. Moreover, the authors address the aspect of reliability by counting the number of 'deceptions', that is, cases in which the true error exceeds the tolerance given by the user. In TESTSET, the most important criteria are: the minimum number of significant correct digits in the numerical solution at the end of the integration interval, the overall number of steps performed by the solver, the number of evaluations of the system function and its Jacobian and the CPU time (on a reference computer). As far as we know, there is no comparable systematics in the verified case although a few single aspects were covered from the theoretical point of view [5].

In the non-verified case, there exists (web-based) software supporting both the user during the choice of the appropriate solver and the developer during the characterization of the product[1]. Aside from the web sites describing verified IVP solvers or general overview sites[2], there are no platforms addressing such topics in the verified case at the moment. Our general (ideal) goal is to provide such a framework which will feature not only the assessment and comparison of different verified solvers but also supply users with means to determine the degree of reliability of their own applications. In this paper, we consider the methodological and theoretical aspects induced by this task.

The paper is structured as follows. In Section 2, we define the problems of interest and introduce a classification scheme for them. In Section 3, we suggest several general criteria for the comparison of verified IVP solvers and discuss the importance of each criterion from the point of view of several application scenarios. In Section 4, the intended presentation form for the comparison is exemplified for two solvers. Finally, we describe the directions for future work in the last section.

[1] e.g. [6], http://num-lab.zib.de/

[2] For example, http://www.cs.utep.edu/interval-comp/

2 Benchmark Problems

Development of a comprehensive set of benchmark problems is one of the fundamental tasks on the way to the development of the comparison framework. We begin this section by defining the class of problems we are generally interested in throughout this paper. Next, we specify a possible classification which might be helpful in showing which solver qualifies best for a given application. Finally, we provide a template for submission of a user problem to the overall set of problems.

2.1 General Problem Formulation

In this paper, we are interested in continuous IVPs for ODEs

$$\dot{x}(t) = f\big(x(t), t\big), \quad x(t_0) \in [x_0] \quad , \tag{1}$$

where $t_0 = 0$ without loss of generality, $t \in [0; t_f] \subset \mathbb{R}$ for some $t_f > 0$, $f \in C^{q-1}(\mathcal{D})$ for some $q > 1$, $\mathcal{D} \subseteq \mathbb{R}^n \times \mathbb{R}$ is open, and $f : \mathcal{D} \mapsto \mathbb{R}^n$. Exact initial values of the variables x are assumed to be inside the (point) interval $[x_0] = [\underline{x}_0 \,;\, \overline{x}_0]$. The function f can depend on bounded parameters p, uncertainties in which are denoted by $[p] = \big[\underline{p} \,;\, \overline{p}\big]$. In this notation, underlined variables denote lower interval bounds (infima) of all components of the corresponding vector, while overlined variables denote upper bounds (suprema). In the case of autonomous ODEs, the notation (1) is abbreviated by $\dot{x}(t) = f(x(t))$.

The problem is discretized on a grid $0 < t_1 < \cdots < t_f = t_m$ with $h_{k-1} = t_k - t_{k-1}$. Denote the solution with the initial condition $x(t_{k-1}) = x_{k-1}$ by $x(t; t_{k-1}, x_{k-1})$ and the set of solutions $\{x(t; t_{k-1}, x_{k-1}) \mid x_{k-1} \in [x_{k-1}]\}$ by $x(t; t_{k-1}, [x_{k-1}])$. The solution of (1) is the set of vectors $[x_k]$ for which the relation $x(t_k; 0, [x_0]) \subseteq [x_k]$, $k = 1, \ldots, m$ holds.

Some of the verified solvers do not discretize the problem. In this case, $[x_k]$ contains the exact solution not only through the point t_k but over the time interval $[t_{k-1}; t_k]$, $k = 1, \ldots, m$. However, we will always consider the former definition for the purposes of uniformity.

A common practice while assessing non-verified solvers is to link the solution and its precision because the former possesses only a limited accuracy. There are also (several different) uses for such tolerances in the verified case [7]. However, they are not as important as for usual numerical solvers because we guarantee the exact solution to lie inside the computed bounds. In this sense, the obtained enclosures are always accurate. What the tolerances do change, is, for example, the break-down time t_{bd} (cf. Criterion 6). That is why it is crucial to unify the definition of tolerances for all considered solvers and to convert varying definitions to the unified one for comparison.

2.2 Classification of Benchmark Problems

The proposed classification is visualized in Fig. 1. As already mentioned, we consider only IVPs for ODEs in this paper[3]. Besides, we cover only the class of

[3] Further practically important tasks would be boundary value problems, differential-algebraic equations or partial differential equations.

non-stiff problems **P.I** from the two main classes **P.I** and **P.II** here, the latter referring to stiff and moderately stiff systems. This classification seems reasonable because of [2],[4],[5],[6]. In both **P.I** and **P.II**, there are the following second-level classes: **L** for linear and $\overline{\mathbf{L}}$ for nonlinear systems. Further, we subdivide each of the second-level classes into

A simple problems with analytical solutions (which help to visualize the validity of the solver),
B moderately complicated systems (with respect to dimension, order, coupling structure, types of nonlinearities) and
C complicated systems.

Even in the linear case, the third-level classes **A**-**C** will be helpful to distinguish between simple and more complicated systems. For example, verified integration of high-dimensional, strongly coupled systems with uncertainties is in general much more demanding than the integration of strongly decoupled or low-dimensional ones. For the class **P.II**, an intermediate subdivision between $\mathbf{L}/\overline{\mathbf{L}}$ and $\mathbf{A}/\mathbf{B}/\mathbf{C}$ levels might become necessary. Important distinguishing characteristics would be the stability of the system, real/non-real eigenvalues as well as linear/nonlinear coupling.

In each of the third-level classes, we differentiate between problems with and without uncertainties (classes **U** and $\overline{\mathbf{U}}$, respectively). Types of uncertainties which can be considered in class **U** are, for example, not exactly known initial values and parameter uncertainties.

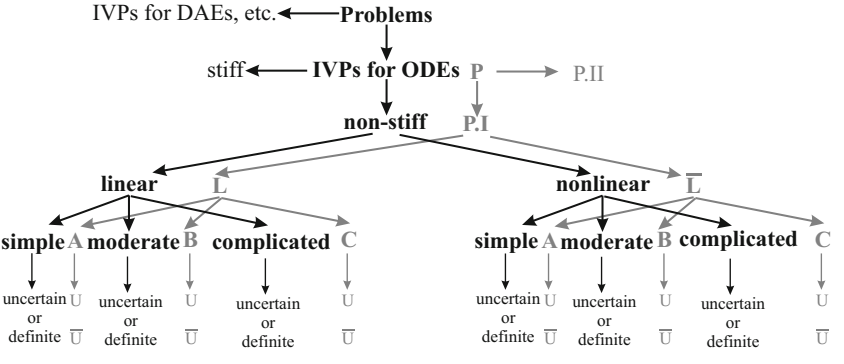


Fig. 1. Classification of benchmark problems for verified IVP solvers.

The problems inside each class are denoted by arabic numbers. In the following, we provide an example problem for each of the classes mentioned above. The notation $\mathbf{P.I.L.A.\overline{U}.1}$, for instance, means the first example in the class of non-stiff, linear, simple problems without uncertainties (cf. Fig. 1).

P.I.L.A.$\overline{\text{U}}$.1 The problem A1 from [2]: $\dot{x} = -x$, $x(0) = 1$, $x(t) = e^{-t}$.
P.I.L.A.U.1 $\dot{x} = -a \cdot x$, $x(0) = 1$, $x(t) = e^{-a \cdot t}$, $a \in [-2; -1]$.

P.I.L.B.$\overline{\text{U}}$.1 B2 from [2]: $\dot{x} = Ax$, $x(0) = [2 \quad 0 \quad 1]^{\mathrm{T}}$, $A = \begin{pmatrix} -1 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -1 \end{pmatrix}$.

P.I.L.B.U.1 As above but with $x(0) \in [[2 - \varepsilon; 2 + \varepsilon] \quad 0 \quad 1]^{\mathrm{T}}$, $\varepsilon = 0.5$.
P.I.L.C.$\overline{\text{U}}$.1 The problem C4 from [2]: $\dot{x} = Ax$, $x(0) = [1 \quad 0 \ldots 0]^{\mathrm{T}}$, $(A)_{51 \times 51}$ is a triple diagonal matrix with -2 on the main diagonal and 1 on the two parallel diagonals.
P.I.L.C.U.1 As above but with $x(0) \in [1 \quad 0 \ldots 0]^{\mathrm{T}} \pm [\varepsilon \quad 0 \ldots 0]^{\mathrm{T}}$, $\varepsilon = 0.5$.
P.I.$\overline{\text{L}}$.A.$\overline{\text{U}}$.1 The problem A2 from [2]: $\dot{x} = -x^3/2$, $x(0) = 1$, $x(t) = 1/\sqrt{t+1}$.
P.I.$\overline{\text{L}}$.A.U.1 As above but with $x(0) \in [\underline{x}_0; \overline{x}_0] = [0.5; 1.5]$, $x(t) \in 1/\sqrt{t + [x_0]^{-2}}$.

P.I.$\overline{\text{L}}$.B.$\overline{\text{U}}$.1 The problem B1 from [2]: $\begin{cases} \dot{x}_1 = 2(x_1 - x_1 x_2), & x_1(0) = 1 \\ \dot{x}_2 = -(x_2 - x_1 x_2), & x_2(0) = 3 \end{cases}$.

P.I.$\overline{\text{L}}$.B.U.1 As above but with $x_1(0) \in [1 - \varepsilon; 1 + \varepsilon]$, $\varepsilon = 0.1$.
P.I.$\overline{\text{L}}$.C.$\overline{\text{U}}$.1 C5 from [2] (the five body problem).
P.I.$\overline{\text{L}}$.C.U.1 The problem C5 as above but with $k_2 \in [2.950; 2.951]$ (the solar gravitational constant).

This classification can be revised after the first testing is done. Groups of problems for which solvers do not show different behavior should be merged. That is, if $C_Q(me_k, pr_i) = C_Q(me_k, pr_j)$ holds for the problem groups pr_i, pr_j, the solver method me_k for each k, and the criterion (cost) C_Q for each Q, then pr_i can be merged with pr_j. Vice versa, if the problems inside of a problem class induce different overall behavior from each solver, this problem class can be split.

The examples above are, of course, only the representatives of each class that should consist of many more of them.

2.3 Template for Problems' Description

In Section 2.2, we classified some basic benchmark problems. However, the users should be encouraged to submit their own problems to the overall framework. For the description of new problems, a format similar to that of [4] can be chosen. At first, the mathematical formulation of each system should be given, including initial values, dimension, exact solution if known, and (uncertain) parameters. An important issue is to specify the desired time horizon $[0\,;\, t_f]$ for which the verified simulation has to be performed. Next, the origins of the problem should be briefly described, which includes the sources of uncertainty in the problem. It might be useful to write down if the problem is chaotic or not, asymptotically stable, neutrally stable or unstable, and constructed or real-life. Preferably, a physical motivation should be given for the chosen values of t_f. Finally, statistics about the numerical solution of the problem based on criteria from Section 3 should be gathered using a specialized program.

3 Criteria for the Comparison of Verified IVP Solvers

The development of the appropriate criteria set for comparing different verified solvers is as important as the development of the set of benchmark problems, but more demanding. It includes definition of quality measures that characterize the solvers and highlight their advantages and drawbacks. In the verified case, we do not have to worry about the *reliability* of the software, which is a priori proven to produce the enclosure of the exact result. That is, $x^{true}(t_k)$ lies inside $[x_k]$. One of the primary concerns is to quantify the *overestimation* produced by the solver, that is, the degree of pessimism the solver adds to the computed upper and lower bounds on the true solution, $\mathrm{diam}([x_k] - x^{true}(t_k))$. In this section, we discuss several such measures inside the set of possible comparison criteria. Afterwards, we classify the proposed criteria with respect to their relevance to a given application scenario using several examples.

3.1 Discussion of General Criteria

The following criteria seem essential while comparing or assessing verified IVP solvers. They are not ordered according to importance or any other attribute. Their sequence is arbitrary because each of them is more or less important depending on the task at hand (cf. Section 3.2). Some of the information required by these criteria should be provided by the developers of the verified software themselves. That includes interfaces for obtaining, for example, the number of function evaluations, etc. The rest of the statistics should be gathered by a specialized program.

C1. Number of arithmetic operations at a time step
C2. Number of function/ Jacobian, etc./ inverse matrix evaluations
C3. Overhead
C4. Wall clock time
C5. User CPU time wrt. resulting interval widths
C6. Time to break-down t_{bd} for each solver.
C7. Total number of steps and the number of accepted steps.

Below, we would like to clarify the exact meaning of those of the criteria which are not self-explanatory.

C1: We are interested in the minimum and maximum number of arithmetic operations as well as the typical number of operations for the suggested default settings. Since switching of rounding modes, memory access, and pipelining can significantly overshadow the influence of the number of arithmetic operations, the criterion will not be used directly for the comparison of verified IVP solvers. However, it seems important because it can help the user to choose appropriate settings for a selected program. Ideally, the developers should provide information on how the number of arithmetic operations will change if solver parameters different from the default settings are set (e.g. increased order of the method,

etc.). This can help the user to find a suitable compromise between the achievable enclosure quality and the required effort.

C3: Analogously to [2], we define overhead as the overall user CPU time minus the user CPU time for function evaluations.

C4: The wall clock time should be computed on selected reference environments for a predefined time $[0\,;\,t_h]$, where t_h should preferably be equal to t_f. For solvers using fixed step sizes, the considered time horizon is a multiple of the step size. Analogously to C1, the influence of the most important tuning parameters of a solver is of interest (the minimum and maximum time, typical time for the suggested default settings). Again, this criterion will not be used directly for the comparison, because it does not take into account the quality of the enclosures. However, it seems important since it shows the user what computing times are achievable and can be expected in principle.

C5: We consider the width $w_h = \max_{i=1}^{n} w_{h,i}$ of the resulting enclosure at t_h for a given problem and all the solvers while gathering statistics for a problem. For a given solver, we average w_h for each problem class **A**–**C** without uncertainties. The width w_h is computed for different tolerances and different method orders (if applicable) but otherwise default settings. For each width, the user CPU time required to obtain the corresponding enclosure on a reference machine is plotted analogously to work-precision diagrams (WPD) from [8].

The definition of quality measures has a direct influence on how fair or 'lifelike' the representation of all solvers in WPDs is. In our opinion, such measures should characterize the resulting overestimation. For systems without uncertainties, a width of the enclosure as described above can give an idea about that. A further possible measure can be the upper bound on the global excess as described in [7].

For systems with uncertainties, the width w_h of the guaranteed enclosure itself does not quantify its tightness with respect to the true solution set. One possibility to measure (and detect) overestimation in a verified way is to evaluate constraints derived independently, for example, on the basis of conservation laws if the mathematical model in question describes a mechanical problem [9]. Alternatively, guaranteed bounds for each interval bound $\underline{x}_i^{true}(t)$ and $\overline{x}_i^{true}(t)$, $i = 1, \ldots, n$, can be computed in the following way[4].

First, we compute a guaranteed enclosure $[x(t)] := [\underline{x}(t)\,;\,\overline{x}(t)]$ of the solution to the IVP for the given initial conditions $[x_0]$. After that, we determine guaranteed enclosures $\left[x^{<j>}(t)\right] := \left[\underline{x}^{<j>}(t)\,;\,\overline{x}^{<j>}(t)\right]$, $j = 1, \ldots, L$, for L selected IVPs with the initial conditions $x^{<j>}(0) \in [x_0]$. Alternatively, intervals $\left[x^{<j>}(0)\right]$ can be considered which are determined by subdivision of $[x_0]$. As shown in Fig. 2, left, the exact lower bound $\underline{x}_i^{true}(t)$ and the exact upper bound $\overline{x}_i^{true}(t)$ of the true solution set are then guaranteed to be contained in the intervals

$$\left[\underline{x}_i^{true}(t)\right] := \left[\underline{x}(t)\,;\,\xi_i(t)\right], \quad \text{where} \quad \xi_i(t) := \min_{j=1,\ldots,L} \left\{\overline{x}^{<j>}(t)\right\} \ ,$$

$$\left[\overline{x}_i^{true}(t)\right] := \left[\zeta_i(t)\,;\,\overline{x}(t)\right], \quad \text{where} \quad \zeta_i(t) := \max_{j=1,\ldots,L} \left\{\underline{x}^{<j>}(t)\right\} \ .$$

[4] To keep the exposition simple, we consider only uncertainties in initial conditions.

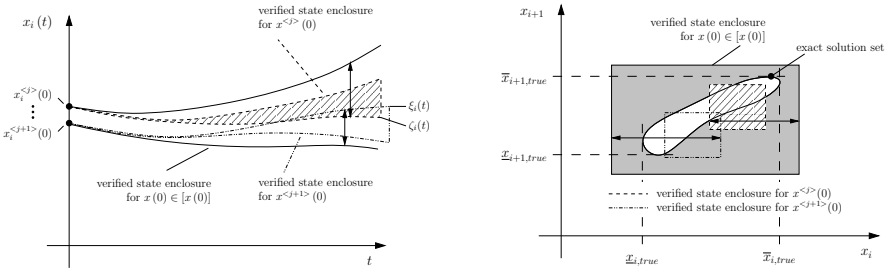


Fig. 2. Verified enclosures for upper and lower bounds of the true solution

For cooperative systems, it is sufficient to consider all vertices $\underline{x}_{0,i}$, $\overline{x}_{0,i}$ of the initial state enclosure $[x_0]$ to obtain *optimal* bounds [10]. Otherwise, interior points should be taken into account. Note that $\xi_i(t) \leq \zeta_i(t)$ not necessarily holds. Furthermore, an interval box which is guaranteed to be contained in the interior of the true solution set cannot be derived from $\xi_i(t)$ and $\zeta_i(t)$ (cf. Fig. 2, right). Of course, this measure can be computed exactly, if the true solution is known analytically (e.g. for the problem **P.I.L.A.U.1**).

C6: Here, we take down the point t_{bd}, after which it is no longer possible to obtain the verified result. The tightness of bounds at t_{bd} is also of interest.

3.2 Applications versus Importance of Each Comparison Criterion

Depending on the kind of practical task at hand, all criteria can be classified into primary and secondary. For simulation of dynamical systems with *numerical* modeling software[5], criterion C2 is of primary importance. If we want to compute verified but rough *a priori enclosures fast*, the major criteria are C4 and C5, whereas C6 is not significant. This type of enclosures might become necessary, for example, in parameter estimation and for state observers with large sampling periods. Sometimes, an *offline* verification of mathematical system models is necessary, for instance, for sensitivity analysis of systems to their parameters, verification of functionality and safety, or optimization of dynamical systems. In this case, C6 is important whereas C4, C5 are of a lesser significance.

The future tests with respect to the benchmark problems will show which criteria are crucial for linear/ nonlinear, stable/ unstable or chaotic systems. To take into account the considerations above, we suggest to differentiate between applications using numerical modeling software as well as online and offline applications. Each new submitted task should be integrated into one of the groups as to allow for an easy choice of the most appropriate criteria. The users can be asked to fill out a questionnaire about their application to find the corresponding group automatically. If it does not fit any group, a new group should be created by an expert who would also provide the right choice of criteria.

[5] That is, software that does not produce symbolic expressions for the resulting IVPs.

4 A Presentation Form for the Comparison

We propose to begin the presentation of the comparison results by a summary similar to that from [2] (averaged performance of solvers on classes of problems, cf. Table 1). As an example of a comparison, we analyzed the solvers VALENCIA-IVP[6] and VNODE-LP[7] quasi-manually. So far, we automated the generation of the files with IVPs for each solver from the same source and the subsequent simulation using both of them (with SMARTMOMAPLE [11]). The reference computer was a four processor dual core (Intel Xeon CPU 2.00GHz) under Linux.

Table 1. Summary over nominal/uncertain problems for $t_h = 1$s and default settings

	C4 ($\overline{\text{U}}$)	C5 ($\overline{\text{U}}$)		...	C4 (U)	C5 (U)		...
		time	width			time	width	
VNODE-LP	0.3653	0.3525	$< 10^{-10}$		0.3580	0.3507	1.4043	
VALENCIA-IVP	28.7393	28.6078	0.0261		24.5238	24.1897	6.4190	

Next, the user might consult the detailed statistics about each benchmark problem. They should include WPDs similar to that in Fig. 3, left. Besides, details on each solver are necessary. Here again, analogously to [4], general information about each solver should be provided, that is, the name, the author(s), the date of the first release, the programming language, availability, web links, etc. The theory should be briefly explained and implementation details given including special options and features, if necessary. This should be followed by showing how to solve simple examples with the given program. Detailed performance statistics for the benchmark problems should include WPDs analogous to Fig. 3, right. For our toy comparison, we chose to use step sizes 0.02, 0.002 and 0.0002 in VALENCIA-IVP and orders 15, 20 and 25 in VNODE-LP (as recommended by the authors).

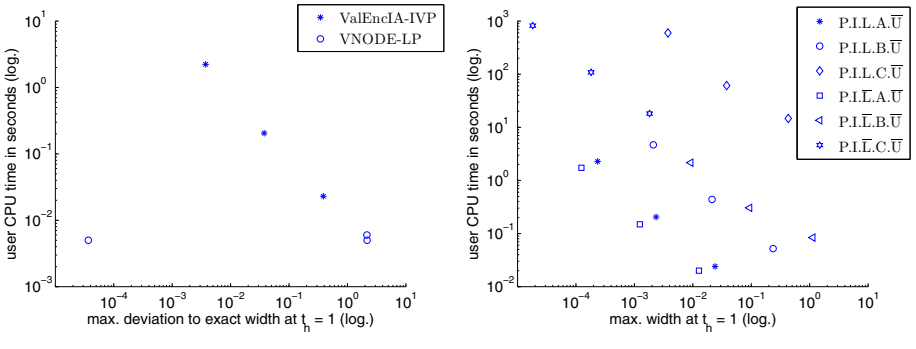


Fig. 3. WPDs for the problem **P.I.L.A.U.1**, left, and VALENCIA-IVP, right

[6] http://valencia-ivp.com/, basic version without exponential state enclosures.
[7] http://www.cas.mcmaster.ca/~nedialk/vnodelp/

From the table and the diagrams, it is possible to induce that for the given problems, VNODE-LP is faster and in most cases less affected by overestimation than the basic version of VALENCIA-IVP.

5 Conclusion

In this paper, we presented a possible framework for comparison of verified IVP solvers. It included a set of problems and a set of criteria that allowed us to characterize the solvers empirically. This work provides a basis for the corresponding software and a web-based platform planned to be implemented in the future.

We would like to thank N.S. Nedialkov and M.A. Stadtherr for their valuable suggestions. This work is funded by the DFG (German Research Foundation).

References

1. Rauh, A., Auer, E., Hofer, E.P., Luther, W. (eds.): Special Issue of the International Journal of Applied Mathematics and Computer Science AMCS, Verified Methods: Applications in Medicine and Engineering, vol. 19(3). University of Zielona Góra Press (2009)
2. Hull, T.E., Enright, W.H., Fellen, B.M., Sedgwick, A.E.: Comparing Numerical Methods for Ordinary Differential Equations. SIAM Journal on Numerical Analysis 9(4), 603–637 (1972)
3. Enright, W.H., Hull, T.E., Lindberg, B.: Comparing Numerical Methods for Stiff Systems of O.D.E:s. BIT Numerical Mathematics 15, 10–48 (1975)
4. Mazzia, F., Iavernaro, F.: Test Set for Initial Value Problem Solvers. Technical Report 40, Department of Mathematics, University of Bari, Italy (2003), http://pitagora.dm.uniba.it/~testset/
5. Neher, M., Jackson, K., Nedialkov, N.: On Taylor Model Based Integration of ODEs. SIAM Journal on Numerical Analysis 45(1), 236–262 (2007)
6. Hall, G., Enright, W., Hull, T., Sedgwick, A.: DETEST: A Program For Comparing Numerical Methods For Ordinary Differential Equations. Technical Report 60, Dept. of Computer Science and Technology, Univ. of Toronto, Toronto (1973)
7. Nedialkov, N.: Computing Rigorous Bounds on the Solution of an Initial Value Problem for an Ordinary Differential Equation. PhD thesis, Department of Computer Science, University of Toronto, Toronto, Canada (1999)
8. Hairer, E., Wanner, G.: Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems, 2nd revised edn. Springer Series in Comput. Mathematics, vol. 14. Springer, Heidelberg (1996)
9. Freihold, M., Hofer, E.P.: Derivation of Physically Motivated Constraints for Efficient Interval Simulations Applied to the Analysis of Uncertain Dynamical Systems. Special Issue of the Intl. Journal of Applied Mathematics and Computer Science AMCS 19(3), 485–499 (2009)
10. Angeli, D., Sontag, E.: Monotone Control Systems. IEEE Transactions on Automatic Control 48(10), 1684–1698 (2003)
11. Reinke, I.: Teilautomatische Aufbereitung von formalen Mehrkörpermodellen zur numerischen Verifikation. Master's thesis, University of Duisburg-Essen (2008)

Fuzzy Solution of Interval Nonlinear Equations

Ludmila Dymova

Institute of Comp. & Information Sci., Czestochowa University of Technology,
Dabrowskiego 73, 42-200 Czestochowa, Poland
dymowa@icis.pcz.pl

Abstract. In [10,12], a new concept of interval and fuzzy linear equations solving based on the generalized procedure of interval extension called "interval extended zero" method has been proposed. The central for this approach is the treatment of "interval zero" as an interval centered around 0. It is shown that such proposition is not of heuristic nature, but is a direct consequence of interval subtraction operation. It is shown that the resulting solution of interval linear equation based on the proposed method may be naturally treated as a fuzzy number. In the current report, the method is extended to the case of nonlinear interval equations. It is shown that opposite to the known methods, a new approach makes it possible to get both the positive and negative solutions of quadratic interval equation.

Keywords: interval nonlinear equation, fuzzy solution, interval zero.

1 Introduction

The problem of interval or fuzzy nonlinear equations is now open since in the literature, there are no universal methods proposed for solving such equations. Although many different numerical methods were proposed for solving interval and fuzzy equations including such complicated as Neural Net solutions [4,5] and fuzzy extension of Newton's method in [1,2], only particular solutions valid in specific conditions were obtained. For example, in [1,3], only the positive root of the quadratic fuzzy equation was obtained, although negative solution can exist too. In our opinion, the root of such problems is the interpretation of the interval and fuzzy extensions. It is known that the equations $F(X) - B = 0, F(X) = B$, where B is an interval or fuzzy value, $F(X)$ is some interval or fuzzy function, are not equivalent ones. Moreover, the main problem is that the conventional interval extension (and the fuzzy as well) of usual equation, which leads to the interval or fuzzy equation such $F(X) - B = 0$ is not a correct procedure. Less problems we meet when dealing with interval or fuzzy equation in form $F(X) = B$, but in many cases its roots are inverted intervals, i.e., such that $\overline{x} < \underline{x}$. To alleviate these problems in the case of linear interval and fuzzy equations, in [10,12] we proposed a new "interval extended zero" method. It was presented rather as useful heuristic [7], which make it possible to solve the system of linear interval equations. In [11], we have used this heuristic for the solution of

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 418–426, 2010.

the specific nonlinear financial problem in the fuzzy setting. Therefore, a more general presentation of this method applied to the solution of nonlinear interval equations seems to be reasonable. In the current report, we show that "interval extended zero" method may be successfully used for solving nonlinear interval equations. Using the same example as in [1,3], we get not only the positive fuzzy solution, but the negative too. The rest of the paper is set out as follows. In Section 2, to clarify the origins of the proposed approach, we recall briefly the basics of "interval extended zero" method in the case of linear interval equations. Section 3 is devoted to presentation of "interval extended zero" method in the case of quadratic interval equation. Section 4 concludes with some remarks.

2 The Basics of "Right Hand Side" Problem and the Solution of Interval Linear Equation

An important methodological problem of interval equations solution, is what we name "interval equation's right hand side problem" [10,12] . Suppose there exists some basic, non-interval algebraic equation

$$f(x) = 0. \tag{1}$$

Its natural interval extension can be obtained by replacement of its variables with interval ones and all arithmetic operations with relevant interval operations. As a result we get an interval equation $[f]([x]) = 0$. Observe that this equation is senseless because its left part represents an interval value, whereas the right part is the non-interval degenerated zero. Obviously, if $[f](x) = [\overline{f}, \underline{f}]$, then equation $[f]([x]) = 0$ is true only when $\overline{f} = \underline{f}$. It is easy to show that the equation $[f]([x]) = 0$, in general, can be verified only for an inverted interval $[x]$, i.e., when $\overline{x} < \underline{x}$. That is why, let us turn to the problem from another point of view. Formally, when extending equation Eq. (1) one obtains not only interval on its left hand side, but interval zero on the right hand side, and in general case, this interval zero cannot be degenerated interval $[0, 0]$. Strictly speaking, in the framework of conventional interval analysis, any interval extension of Eq. (1) is not a correct operation since we obtain an interval mathematical expression only in the left hand side of equation, whereas in its right hand side the usual zero integer is not changed. In our opinion, the root of problem is that conventional approach to interval extension does not involve an operation we call "interval zero extension". In other words, we propose an operation of "interval zero extension" to obtain an "interval zero" in the right side of extended Eq. (1). Since "interval zero" is not a degenerated interval, such approach makes it possible to solve the problem of correct interval extension of Eq. (1). First of all, what is "interval zero" ? In conventional interval analysis, it is usually assumed that any interval containing zero may be considered as "interval zero". This is a satisfactory definition to suppress the division by zero in conventional interval arithmetic, but for our purposes a more restrictive definition is needed. Let us look to this problem from another point of view. Without loss of generality, we

can define the degenerated (usual) zero as the result of operation $a-a$, where a is any real valued number or variable. Hence, in a similar way we can define an "interval zero" as the result of operation $[a]-[a]$, where $[a]$ is an interval. It is easy to see that for any interval $[a]$ we get $[\underline{a},\overline{a}]-[\underline{a},\overline{a}]=[\underline{a}-\overline{a},\overline{a}-\underline{a}]$. Therefore, in any case the result of interval subtraction $[a]-[a]$, is an interval centered around 0. Thus, if we want to treat a result of subtraction of two identical intervals as "interval zero", then the most general definition of such "zero" will be "interval zero is an interval symmetrical with respect to 0". It must be emphasized that introduced definition says nothing about the width of "interval zero".

Let us consider the linear equation

$$ax-b=0. \tag{2}$$

When extending equation such Eq. (2) with previously unknown values of variables in the left hand side, only what we can say about right hand side is that it should be interval symmetrical with respect to 0 with not defined width. Hence, as the result of interval extension of Eq. (2) in general case we get

$$[\underline{a},\overline{a}][\underline{x},\overline{x}]-[\underline{b},\overline{b}]=[-y,y]. \tag{3}$$

In fact, the right hand side of Eq. (3) is some interval centered around zero, which can be treated as interval extension of the right hand side of Eq. (2), in other words, as an interval extension of 0. This is the reason for us to call our approach "interval extended zero" method. Of course, the value of y in Eq. (3) is not yet defined and this seems to be quite natural since the values of $\underline{x},\overline{x}$ are also not defined. At first, consider the case of positive interval numbers $[a]$ and $[b]$, i.e., $\underline{a}$, $\overline{a}$,$\underline{b}$, $\overline{b}>0$. Then from Eq. (3) we get

$$\begin{cases} \underline{ax}-\overline{b}=-y, \\ \overline{ax}-\underline{b}=y. \end{cases} \tag{4}$$

Finally, from Eq. (4) we obtain only one linear equation with two unknown variables $\underline{x}$ and $\overline{x}$:

$$\underline{ax}+\overline{ax}-\underline{b}-\overline{b}=0. \tag{5}$$

If there are some restrictions on the values of unknown variables $\underline{x}$ and $\overline{x}$, then Eq. (5) with these restrictions may be considered as the so called Constraint Satisfaction Problem (CSP) [6] and its interval solution may be obtained. The first restriction on the variables $\underline{x}$ and $\overline{x}$ is a solution of Eq. (5) assuming $\underline{x}=\overline{x}$. In this degenerated case we get the solution of Eq. (5) as $x_m=\frac{\underline{b}+\overline{b}}{\underline{a}+\overline{a}}$. It is easy to see that x_m is an upper bound for $\underline{x}$ and a lower bound for $\overline{x}$. The natural low bound for $\underline{x}$ and upper bond for $\overline{x}$ may be defined using basic definitions of interval arithmetic [8,9] as $\underline{x}=\frac{\underline{b}}{\overline{a}}$, $\overline{x}=\frac{\overline{b}}{\underline{a}}$. Thus, we have $[\underline{x}]=[\frac{\underline{b}}{\overline{a}},x_m]$ and $[\overline{x}]=[x_m,\frac{\overline{b}}{\underline{a}}]$. These intervals can be narrowed taking into account Eq. (5), which in the spirit of CSM is treated as the restriction. It is clear that the right bound of $\underline{x}$ and left bound of $\overline{x}$, i.e., x_m, can not be changed as they present the

degenerated (crisp) solution of (5). So let us focus of the left bound of $\underline{x}$ and right bound of $\overline{x}$. From (5) we have

$$\underline{x} = \frac{\underline{b} + \overline{b} - \overline{a}\overline{x}}{\underline{a}}, \overline{x} \in [x_m, \frac{\overline{b}}{\underline{a}}], \overline{x} = \frac{\underline{b} + \overline{b} - \underline{a}\underline{x}}{\overline{a}}, \underline{x} \in [\frac{\underline{b}}{\overline{a}}, x_m]. \tag{6}$$

Obviously, when $\overline{x}$ is maximal, i.e., $\overline{x} = \frac{\overline{b}}{\underline{a}}$, we get the minimal value of $\underline{x}$, i.e., $\underline{x}_{\min} = \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\overline{a}\overline{b}}{\underline{a}^2}$. Similarly, from (6) we get the maximal value of $\overline{x}$, i.e., $\overline{x}_{\max} = \frac{\underline{b}+\overline{b}}{\overline{a}} - \frac{\underline{a}\underline{b}}{\overline{a}^2}$. Since it is possible that $\underline{x}_{\min} < \frac{\underline{b}}{\overline{a}}$ and $\overline{x}_{\max} > \frac{\overline{b}}{\underline{a}}$, we get the following interval solution:

$$[\underline{x}] = \left[\underline{x}_{\max}, \frac{\underline{b} + \overline{b}}{\underline{a} + \overline{a}}\right], [\overline{x}] = \left[\frac{\underline{b} + \overline{b}}{\underline{a} + \overline{a}}, \overline{x}_{\min}\right], \tag{7}$$

where $\underline{x}_{\max} = \max\left(\frac{\underline{b}}{\overline{a}}, \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\overline{a}\overline{b}}{\underline{a}^2}\right)$, $\overline{x}_{\min} = \min\left(\frac{\overline{b}}{\underline{a}}, \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\underline{a}\underline{b}}{\overline{a}^2}\right)$. It is important that in the framework of CSP, the following relations between $\underline{x}_{\max}$ and $\overline{x}_{\min}$ should be fulfilled in calculations: if $\underline{x}_{\max} = \frac{\underline{b}}{\overline{a}}$, then $\overline{x}_{\min} = \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\underline{a}\underline{b}}{\overline{a}^2}$; if $\overline{x}_{\min} = \frac{\overline{b}}{\underline{a}}$, then $\underline{x}_{\max} = \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\overline{a}\overline{b}}{\underline{a}^2}$. Expressions (7) define all possible solutions of Eq. (3). The values of $\overline{x}_{min}$, $\underline{x}_{max}$ constitute an interval which produce the widest interval zero after substitution of them in Eq. (4). In other words, the maximum interval solution's width $w_{\max} = \overline{x}_{\min} - \underline{x}_{\max}$ corresponds to the maximum value of y: $y_{\max} = \frac{\overline{a}\overline{b}}{\underline{a}} - \underline{b}$. Substitution of degenerated solution $\underline{x} = \overline{x} = x_m$ in Eq. (3) produces the minimum value of y: $y_{min} = \frac{\overline{a}\cdot\overline{b} - \underline{a}\cdot\underline{b}}{\underline{a}+\overline{a}}$. It is clear that for any permissible solution $\underline{x}' > \underline{x}_{max}$ we have corresponding $\overline{x}' < \overline{x}_{min}$, for each $\underline{x}'' > \underline{x}'$ the inequalities $\overline{x}'' < \overline{x}'$ and $y'' < y'$ take place. Thus, the formal interval solution (7) factually represents the continuous set of nested interval solutions of Eq.(3). Hereinafter, we show that this set of interval solutions can be naturally interpreted as a fuzzy number. We can see that values of y characterize the closeness of right hand side of Eq. (3) to degenerated zero and minimum value y_{min} is defined exclusively by interval parameters $[a]$ and $[b]$. Hence, the values of y may be considered, in a certain sense, as a measure of interval solution's uncertainty caused by the initial uncertainty of Eq. (3). Therefore we introduce

$$\alpha = 1 - \frac{y - y_{min}}{y_{max} - y_{min}}, \tag{8}$$

which may be treated as a certainty degree of interval solution of Eq. (3). We can see that α rises from 0 to 1 with decreasing of interval's width from maximum value to 0, i.e., with increasing of solution's certainty. Consequently, the values of α may be treated as labels of α-cuts representing some fuzzy solution of Eq. (3). Finally, we obtain a solution in form of triangular fuzzy number

$$\tilde{x} = \left\{\underline{x}_{\max}, \frac{\underline{b} + \overline{b}}{\underline{a} + \overline{a}}, \overline{x}_{\min}\right\}. \tag{9}$$

In a similar way, the fuzzy solutions of Eq. (3) were obtained for other placements of intervals $[a]$ and $[b]$ (see [10,12]). Obviously, we can assume the support of obtained fuzzy number to be a solution of analyzed problem. Such a solution may be treated as the "pessimistic" one since it corresponds to the lowest α-cuts of resulting fuzzy value. We use here the word "pessimistic" to emphasize that this solution is charged with the largest imprecision as it is obtained in the most uncertain conditions possible on the set of considered α -cuts. On the other hand, it seems natural to utilize all additional information available in the fuzzy solution. We can reduce the resulting fuzzy solution to the interval solution using well known defuzzification procedures. In our case, defuzzified left and right boundaries of the solution can be represented as

$$\underline{x}_{def} = \frac{\int_0^1 \underline{x}(\alpha)d\alpha}{\int_0^1 d\alpha}, \overline{x}_{def} = \frac{\int_0^1 \overline{x}(\alpha)d\alpha}{\int_0^1 d\alpha} \tag{10}$$

For example, in the case of $[a], [b] > 0$, from (3) and (8) we get the expressions for $\underline{x}(\alpha)$ and $\overline{x}(\alpha)$. Substituting them into (10) we obtain $\underline{x}_{def} = \frac{\overline{b}}{\underline{a}} - \frac{y_{max}+y_{min}}{2\underline{a}}, \overline{x}_{def} = \frac{\underline{b}}{\overline{a}} + \frac{y_{max}+y_{min}}{2\overline{a}}$.

It is easy to prove that obtained interval $[\underline{x}_{def}, \overline{x}_{def}]$ is included into support interval of initial fuzzy solution, i.e., $[\underline{x}_{max}, \overline{x}_{min}]$. It is shown in [10,12] that the proposed method provides the considerable reducing of resulting interval's length in comparison with that obtained using conventional interval arithmetic rules.

3 Fuzzy Solution of Nonlinear Interval Equation

The general approach described in previous Section can be adapted for solving nonlinear equations. The method we develop in this Section can be applied for solving a wide range of nonlinear interval equations if some initial restrictions on the solution's bounds are known. Nevertheless to present our method more transparent, we consider the well known example of quadratic fuzzy equation [1,3] that factually can be treated as the test task:

$$ax^2 + bx - c = 0, \tag{11}$$

where a, b, c are intervals.

Although in [1,3], the parameters a, b and c were presented by trapezoidal and triangular fuzzy numbers, here for our purpose it is enough to use only their supports: $a = [3, 5]$, $b = [1, 3]$, $c = [1, 3]$.

Whereas it is stated in [1,3] that Eq. (11) with these parameters have no negative fuzzy root, we shall obtain such root. In the spirit of "interval extended zero" method described in Section 2, we represent Eq. (11) in the following form:

$$[\underline{a}, \overline{a}][\underline{x}, \overline{x}]^2 + [\underline{b}, \overline{b}][\underline{x}, \overline{x}] - [\underline{c}, \overline{c}] = [-y, y], \tag{12}$$

where y is undefined parameter (see Section 2). Using conventional interval arithmetic rules from Eq.(12) we get

$$[\underline{a}\underline{x} + \underline{b}, \overline{a}\overline{x} + \overline{b}][\underline{x}, \overline{x}] - [\underline{c}, \overline{c}] = [-y, y]. \tag{13}$$

Firstly consider the case of positive interval root of Eq.(13) ,i.e., $\underline{x}, \overline{x} > 0$. Then from (13) we obtain

$$\underline{a}\underline{x}^2 + \underline{b}\underline{x} - \overline{c} = -y, \overline{a}\overline{x}^2 + \overline{b}\overline{x} - \underline{c} = y. \tag{14}$$

The sum of Eqs.(14) results in

$$\underline{a}\underline{x}^2 + \underline{b}\underline{x} - \overline{c} + \overline{a}\overline{x}^2 + \overline{b}\overline{x} - \underline{c} = 0. \tag{15}$$

As in the case of real valued a, b, c, the positive root of (11) is presented by the expression $x = \frac{-b+\sqrt{b^2+4ac}}{2a}$, the "natural restrictions" on the positive interval solution of (11) can be represented as

$$x_{min} = \frac{-\overline{b} + \sqrt{\underline{b}^2 + 4\underline{a}\underline{c}}}{2\overline{a}}, x_{max} = \frac{-\underline{b} + \sqrt{\overline{b}^2 + 4\overline{a}\overline{c}}}{2\underline{a}}. \tag{16}$$

Similar to the case of linear interval equation (see Section 2) we consider the real valued (degenerated) solution of Eq.(15), x_m, as the natural top bound for positive $\underline{x}$, i.e., $\underline{x} \le x_m$ and bottom bound for positive $\overline{x}$, i.e., $x_m \le \overline{x}$. For the case of $\underline{x} = \overline{x} = x_m$ from (15) we get

$$x_m = \frac{-(\underline{b} + \overline{b}) + \sqrt{(\underline{b} + \overline{b})^2 + 4(\underline{a} + \overline{a})(\underline{c} + \overline{c})}}{2(\underline{a} + \overline{a})}. \tag{17}$$

Eq.(15) with described above restrictions $x_{\min} \le \underline{x} \le x_m, x_m \le \overline{x} \le x_{\max}$ is a typical Constraint Satisfaction Problem [6] and its interval solution can be obtained. From Eq.(15) we get the expressions

$$\underline{x} = \underline{f}(\overline{x}) = \frac{-\underline{b}+\sqrt{\underline{b}^2+4\underline{a}(\underline{c}+\overline{c}-\overline{a}\overline{x}^2-\overline{b}\overline{x})}}{2\underline{a}}, \overline{x} = \overline{f}(\underline{x}) = \frac{-\overline{b}+\sqrt{\overline{b}^2+4\overline{a}(\underline{c}+\overline{c}-\underline{a}\underline{x}^2-\underline{b}\underline{x})}}{2\overline{a}}.$$

Generally, the interval solution of the above constraint satisfaction problem can be represented as follows:

$$[\underline{x}] = [x_{\min}, x_m] \cap [\underline{x}_1^*, \underline{x}_2^*], [\overline{x}] = [x_m, x_{\max}] \cap [\overline{x}_1^*, \overline{x}_2^*], \tag{18}$$

where $\underline{x}_1^* = \min \underline{f}(\overline{x})$, $\underline{x}_2^* = \max \underline{f}(\overline{x})$ $(x_m \le \overline{x} \le x_{\max})$; $\overline{x}_1^* = \min \overline{f}(\underline{x})$, $\overline{x}_2^* = \max \overline{f}(\underline{x})$ $(x_{\min} \le \underline{x} \le x_m)$.

It is easy to see that in our case

$$\underline{x}_1^* = \frac{-\underline{b}+\sqrt{\underline{b}^2+4\underline{a}(\underline{c}+\overline{c}-\overline{a}x_{\max}^2-bx_{\max})}}{2\underline{a}}, \underline{x}_2^* = \frac{-\underline{b}+\sqrt{\underline{b}^2+4\underline{a}(\underline{c}+\overline{c}-\overline{a}x_m^2-bx_m)}}{2\underline{a}},$$
$$\overline{x}_1^* = \frac{-\overline{b}+\sqrt{\overline{b}^2+4\overline{a}(\underline{c}+\overline{c}-\underline{a}x_m^2-\underline{b}\underline{x}_m)}}{2\overline{a}}, \overline{x}_2^* = \frac{-\overline{b}+\sqrt{\overline{b}^2+4\overline{a}(\underline{c}+\overline{c}-\underline{a}x_{\min}^2-\underline{b}\underline{x}_{\min})}}{2\overline{a}}.$$

It is clear that Exp. (18) leads to the interval solution

$$[\underline{x}] = [\underline{x}_{\min}, \underline{x}_{\max}], [\overline{x}] = [\overline{x}_{\min}, \overline{x}_{\max}], \tag{19}$$

where $\underline{x}_{\min} = \max(x_{\min}, \underline{x}_1^*)$, $\underline{x}_{\max} = \min(x_m, \underline{x}_2^*)$, $\overline{x}_{\min} = \max(x_m, \overline{x}_1^*)$, $\overline{x}_{\max} = \min(x_{\max}, \overline{x}_2^*)$.

In the considered numerical example ($a = [3, 5]$, $b = [1, 3]$, $c = [1, 3]$), we have obtained $\underline{x}_{\max} = \overline{x}_{\min} = x_m$. As in the linear case (see Section 2), substituting the widest possible interval solution $[\underline{x}_{\min}, \overline{x}_{\max}]$ into Eq. (13) we get the maximal value of y, i.e. $y_{\max}$, and substituting in this equation the shortest possible solution $[\underline{x}_{\max}, \overline{x}_{\min}] = [x_m, x_m]$ we obtain $y_{\min}$. As in the linear case, the formal interval solution (19) factually represents the continuous set of nested interval solutions of Eq.(13) and we can use the Exp.(8) to calculate the values of y on the α-cuts. For α rising from 0 to 1 using (8) we get the value of y and substituting them into (14) we obtain the set of interval solutions $[\underline{x}, \overline{x}]_\alpha$ on the corresponding α-cuts. As the result, the positive fuzzy solution presented in Fig.1 has been obtained.

Using the proposed method, the negative root ($\underline{x}, \overline{x} < 0$) of fuzzy Eq.(13) can be obtained as well. For this case we get the following set of expressions:

$$\underline{a}\overline{x}^2 + \overline{b}\underline{x} - \overline{c} = -y, \overline{a}\underline{x}^2 + \underline{b}\overline{x} - \underline{c} = y. \tag{20}$$

$$\underline{a}\overline{x}^2 + \overline{b}\underline{x} - \overline{c} + \overline{a}\underline{x}^2 + \underline{b}\overline{x} - \underline{c} = 0. \tag{21}$$

$$x_m = \frac{-(\underline{b} + \overline{b}) - \sqrt{(\underline{b} + \overline{b})^2 + 4(\underline{a} + \overline{a})(\underline{c} + \overline{c})}}{2(\underline{a} + \overline{a})}, \tag{22}$$

$$x_{min} = \frac{-\overline{b} - \sqrt{\overline{b}^2 + 4\overline{a}\overline{c}}}{2\underline{a}}, x_{max} = \frac{-\underline{b} - \sqrt{\underline{b}^2 + 4\underline{a}\underline{c}}}{2\overline{a}}. \tag{23}$$

$$\underline{x} = \underline{f}(\overline{x}) = \frac{-\overline{b} - \sqrt{\overline{b}^2 + 4\overline{a}(\underline{c} + \overline{c} - \underline{a}\overline{x}^2 - \underline{b}\overline{x})}}{2\overline{a}},$$
$$\overline{x} = \overline{f}(\underline{x}) = \frac{-\underline{b} - \sqrt{\underline{b}^2 + 4\underline{a}(\underline{c} + \overline{c} - \overline{a}\underline{x}^2 - \overline{b}\underline{x})}}{2\overline{a}} \tag{24}$$

$$[\underline{x}] = [x_{\min}, x_m] \cap [\underline{x}_1^*, \underline{x}_2^*], [\overline{x}] = [x_m, x_{\max}] \cap [\overline{x}_1^*, \overline{x}_2^*], \tag{25}$$

where $\underline{x}_1^* = \min \underline{f}(\overline{x})$, $\underline{x}_2^* = \max \underline{f}(\overline{x})$ $(x_m \le \overline{x} \le x_{\max})$; $\overline{x}_1^* = \min \overline{f}(\underline{x})$, $\overline{x}_2^* = \max \overline{f}(\underline{x})$ $(x_{\min} \le \underline{x} \le x_m)$.

The numerical algorithm we have used to obtain the negative root is similar to that we have presented above for the positive root. The result is presented in Fig.1.

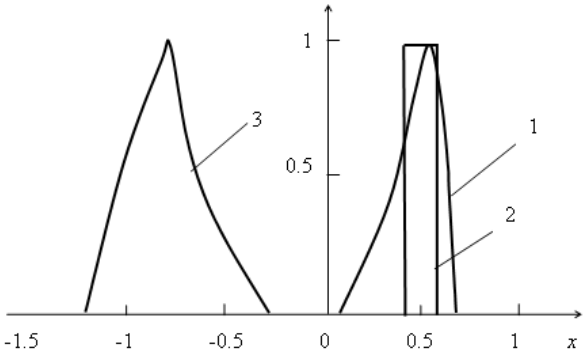


Fig. 1. The fuzzy roots of quadratic interval equation: 1,3-fuzzy solution obtained with use of "interval extended zero" method, 2-interval solution from [1,3]

It is seen that our positive fuzzy solution in the considered example is wider than the interval solution obtained in [1,3]. Nevertheless, it does not mean that the results from [1,3] are more "true" since the methods proposed in [1,3] except the obtaining of negative roots. Besides, our results may be substantially shortened using the reduction of fuzzy solution to an interval one with a help of defuzzification procedure (10).

Summarizing we can say that the proposed method allows us to get positive and negative fuzzy solutions of interval quadratic equations, whereas known approaches do not provide negative solutions. In [11], we have shown that our method can be successfully used for solving more complicated nonlinear problems, but in [11], the method is performed rather as a heuristic approach.

4 Conclusion

The aim of the paper is to present an extension of the so called "interval extended zero" method proposed in [10,12], to the case of nonlinear interval equations. The key idea of this method is the treatment of "interval zero" as an interval symmetrical with respect to 0. It is shown that such approach is a direct consequence of interval subtraction operation. It is shown that the method provides a fuzzy solution of nonlinear interval equations. It is important that opposite to the known approaches, the method makes it possible to get both the positive and negative fuzzy solutions of interval quadratic equation.

References

1. Abbasbandy, S., Asady, B.: Newton's method for solving fuzzy nonlinear equations. Applied Mathematics and Computation 159, 349–356 (2004)
2. Abbasbandy, S.: Extended Newton's method for a system of nonlinear equations by modified Adomian decomposition method. Applied Mathematics and Computation 170, 648–656 (2005)

3. Buckley, J.J., Qu, Y.: Solving linear and quadratic fuzzy equations. Fuzzy Sets and Systems 38, 43–59 (1990)
4. Buckley, J.J., Eslami, E.: Neural net solutions to fuzzy problems: The quadratic equation. Fuzzy Sets and Systems 86, 289–298 (1997)
5. Buckley, J.J., Eslami, E., Hayashi, Y.: Solving fuzzy equations using neural nets. Fuzzy Sets and Systems 86, 271–278 (1997)
6. Cleary, J.C.: Logical Arithmetic. Future Computing Systems 2, 125–149 (1987)
7. Dymova, L., Gonera, M., Sevastianov, P., Wyrzykowski, R.: New method for interval extension of Leontiefs input-output model with use of parallel programming. In: Proc. Int. Conf. on Fuzzy Sets and Soft Computing in Economics and Finance, St. Petersburg, pp. 549–556 (2004)
8. Jaulin, L., Kieffir, M., Didrit, O., Walter, E.: Applied Interval Analysis. Springer, London (2001)
9. Moore, R.E.: Interval analysis. Prentice-Hall, Englewood Cliffs (1966)
10. Sevastjanov, P., Dymova, L.: Fuzzy solution of interval linear equations. In: Proc. of 7th Int. Conf. Paralel Processing and Applied Mathematics, Gdansk, pp. 1392–1399 (2007)
11. Sewastjanow, P., Dymowa, L.: On the Fuzzy Internal Rate of Return. In: Kahraman, C. (ed.) Fuzzy Engineering Economics with Applications, pp. 105–128. Springer, Heidelberg (2008)
12. Sevastjanov, P., Dymova, L.: A new method for solving interval and fuzzy equations: linear case. Information Sciences 17, 925–937 (2009)

Solving Systems of Interval Linear Equations with Use of Modified Interval Division Procedure

Ludmila Dymova, Mariusz Pilarek, and Roman Wyrzykowski

Institute of Comp.& Information Sci., Czestochowa University of Technology,
Dabrowskiego 73, 42-200 Czestochowa, Poland
dymowa@icis.pcz.pl

Abstract. A new approach to interval division based on the concept of "interval extended zero" method [10,11] is proposed. This modified interval devision is used for solving the systems of interval linear equations. The seven known examples are used as an illustration of the method's efficacy. It is shown that a new method provides results close to the so-called maximal inner solution. The method not only allows us to decrease the excess width effect, but makes it possible to avoid the inverted interval solutions too.

Keywords: system of interval linear equations, interval zero method, modified interval devision.

1 Introduction

The system of linear interval equations can be presented as follows

$$Ax = b, \tag{1}$$

where A is an interval matrix, b is an interval vector and x is an interval vector solution. Generally, such a system has no exact solutions, since usually it is not possible to find such interval x for which Ax is exactly equal to the interval b. Nevertheless, the different particular solutions of Eqs.(1)were proposed in the literature. Now the dominant approaches to the solution of linear interval and fuzzy systems are based on the treating of Eqs. (1) as a set of real valued equations whose parameters belong to the corresponding intervals or fuzzy intervals A and b [3]. In this framework, the important are the concepts of the united solution set (USS), its subsets called the tolerable solution set (TSS) and the controllable solution set [3]. The undesirable feature of known approaches to the solution of Eqs. (1) is the so-called excess width effect. Therefore, recently the considerable efforts were made to reduce it with use of various versions of the stationary single-step iteration method proposed by V. Zyuzin [13]. Kupriyanova [4] proved the convergence of this iterative process to the so-called maximal inner solution of problem (1) under special (implicit) restrictions on the input data A,

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 427–435, 2010.

b and on the initial approximation. Markov [6,7] formulated these restrictions explicitly in context of Jacobi type iterative method for the solution of Eqs. (1). It is important that these iterative methods provide the so-called maximal inner solutions, i.e., approximate solutions with minimal excess width effect. Another approach aspiring to reduce the excess width effect in the solution of Eqs. (1) is based on the so-called "interval extended zero" method [10]. In [11], we have used this approach to solve the interval extended Leontief's Input -Output problem [5] without drastic increase of the resulting intervals. It is worthy to note that this problem is characterized by the specific form of matrix A. So the obtained good results can not to be considered as a solid evidence of the method's efficacy. In the current report, we show that "interval extended zero" method can be naturally treated as the modified interval division and use this division for building effective algorithms for solving linear interval systems. To illustrate a new method, we present the results obtained for known examples repeatedly used in the literature as the tests for numerical methods in the interval setting. Comparing our results with those obtained using Markov's method [6,7] and usual interval Gauss elimination procedure we show that the proposed method not only allows us to decrease the excess width effect, but makes it possible to avoid inverted interval solutions too. The rest of the paper is set out as follows. In Section 2, we recall the foundations of "interval extended zero" method and present its interpretation as the modified interval division. Section 3 presents the application of modified interval division to the solution of interval problem (1) and illustrative examples. Section 4 concludes with some remarks.

2 "Interval Extended Zero" Method and Its Interpretation as Modified Interval Division

Let us consider the simplest equation

$$ax - b - 0, \tag{2}$$

where a,b are real values. Its conventional interval extension leads to the interval equation

$$[a][x] - [b] = 0, \tag{3}$$

which seems to be senseless because its left part represents an interval value, whereas the right part is the non-interval degenerated zero. It is easy to show that Eq.(3) can be verified only for an inverted interval $[x]$, i.e., when $\overline{x} < \underline{x}$. That is why, let us turn to the problem from another point of view. Formally, when extending Eq. (2) one obtains not only interval on its left hand side, but interval zero on the right hand side. Generally, this interval zero cannot be degenerated interval $[0, 0]$. Therefore, the concepts of "interval zero extension" and "interval zero" has been proposed in [10]. The operation of "interval zero extension" provides an "interval zero" in the right side of extended Eq. (2). Since "interval zero" is not a degenerated interval, such approach makes it possible to solve the problem of correct interval extension of Eq. (2). The concept of "interval zero"

is based on the following reasoning [10,11]. In conventional interval analysis, it is usually assumed that any interval containing zero may be considered as "interval zero". Let us look to this problem from another point of view. Without loss of generality, we can define the degenerated (usual) zero as the result of operation $a-a$, where a is any real valued number or variable. Hence, in a similar way we can define an "interval zero" as the result of operation $[a]-[a]$, where $[a]$ is an interval. It is easy to see that for any interval $[a]$ we get $[\underline{a},\overline{a}]-[\underline{a},\overline{a}] = [\underline{a}-\overline{a},\overline{a}-\underline{a}]$. Therefore, in any case the result of interval subtraction $[a]-[a]$, is an interval centered around 0. Thus, if we want to treat a result of subtraction of two identical intervals as "interval zero", then the most general definition of such "zero" will be "interval zero is an interval symmetrical with respect to 0". It must be emphasized that introduced definition says nothing about the width of "interval zero".

Thus, when extending Eq. (2) with previously unknown values of variables in the left hand side, only what we can say about the right hand side is that it should be an interval symmetrical with respect to 0 with not defined width. Hence, as the result of interval extension of Eq. (2) in general case we get

$$[\underline{a},\overline{a}][\underline{x},\overline{x}] - [\underline{b},\overline{b}] = [-y, y]. \tag{4}$$

In fact, the right hand side of Eq. (4) is some interval centered around zero, which can be treated as interval extension of the right hand side of Eq. (2),i.e., as an interval extension of 0. The value of y in Eq. (4) is not yet defined since the values of $\underline{x},\overline{x}$ are also not defined. As there are three unknowns ($\underline{x}$,$\overline{x}$ and y) in Eq. (4), it has no real valued solution. But, if there are some constraints on the values of unknown variables $\underline{x}$ and $\overline{x}$, then Eq. (4) with these constraints may be considered as the so-called Constraint Satisfaction Problem (CSP) [1] and its interval solution may be obtained [10,11].

Since Eq. (4) is the interval extension of Eq. (2) which can be presented in algebraically equivalent form $x = \frac{b}{a}$, this solution can be considered also as the result of modified interval devision $[x]_{\,\mathrm{mod}} = \left(\frac{[b]}{[a]}\right)_{\mathrm{mod}}$. Such a treatment of the solution of Eq. (4) appears to be very efficient in the solution of interval linear systems (see Section 3). The modified interval devision can be treated as an alternative to the conventional interval devision: $\underline{x} = \frac{\underline{b}}{\overline{a}}$, $\overline{x} = \frac{\overline{b}}{\underline{a}}$.

At first, let us obtain the solution of Eq. (4) or $[x]_{\,\mathrm{mod}}$ in the case of positive interval numbers $[a]$ and $[b]$, i.e., $\underline{a}$, $\overline{a}$,$\underline{b}$, $\overline{b} > 0$. In this case, from Eq. (4) we get

$$\underline{ax} - \overline{b} = -y, \overline{ax} - \underline{b} = y. \tag{5}$$

From Eq. (5) we obtain only one linear equation with two unknown variables $\underline{x}$ and $\overline{x}$:

$$\underline{ax} + \overline{ax} - \underline{b} - \overline{b} = 0. \tag{6}$$

If there are some constraints on the values of unknown variables $\underline{x}$ and $\overline{x}$, then Eq. (6) with these constraints may be considered as the so-called Constraint Satisfaction Problem (CSP) [1] and its interval solution may be obtained. The

first constraint on the variables $\underline{x}$ and $\overline{x}$ is a solution of Eq. (6) assuming $\underline{x} = \overline{x}$. In this degenerated case we get the solution of Eq. (6) as $x_m = \frac{\underline{b}+\overline{b}}{\underline{a}+\overline{a}}$. It is seen that x_m is the upper bound for $\underline{x}$ and the lower bound for $\overline{x}$. The natural low bound for $\underline{x}$ and upper bond for $\overline{x}$ may be defined using basic definitions of interval arithmetic [8] as $\underline{x} = \frac{\underline{b}}{\overline{a}}$, $\overline{x} = \frac{\overline{b}}{\underline{a}}$. Thus, we have $[\underline{x}] = [\frac{\underline{b}}{\overline{a}}, x_m]$ and $[\overline{x}] = [x_m, \frac{\overline{b}}{\underline{a}}]$. These intervals can be narrowed taking into account Eq. (6), which in the spirit of CSM is treated as a constraint. From (6) we get

$$\underline{x} = \frac{\underline{b} + \overline{b} - \overline{a}\overline{x}}{\underline{a}}, \overline{x} \in [x_m, \frac{\overline{b}}{\underline{a}}], \overline{x} = \frac{\underline{b} + \overline{b} - \underline{a}\underline{x}}{\overline{a}}, \underline{x} \in [\frac{\underline{b}}{\overline{a}}, x_m]. \tag{7}$$

Obviously, when $\overline{x}$ is maximal, i.e., $\overline{x} = \frac{\overline{b}}{\underline{a}}$, we get the minimal value of $\underline{x}$, i.e., $\underline{x}_{\min} = \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\overline{a}\overline{b}}{\underline{a}^2}$. Similarly, from (7) we get the maximal value of $\overline{x}$, i.e., $\overline{x}_{\max} = \frac{\underline{b}+\overline{b}}{\overline{a}} - \frac{\underline{a}\underline{b}}{\overline{a}^2}$. Since it is possible that $\underline{x}_{\min} < \frac{\underline{b}}{\overline{a}}$ and $\overline{x}_{\max} > \frac{\overline{b}}{\underline{a}}$, we get the following interval solution:

$$[\underline{x}] = \left[\underline{x}_{\max}, \frac{\underline{b} + \overline{b}}{\underline{a} + \overline{a}}\right], [\overline{x}] = \left[\frac{\underline{b} + \overline{b}}{\underline{a} + \overline{a}}, \overline{x}_{\min}\right], \tag{8}$$

where $\underline{x}_{\max} = \max\left(\frac{\underline{b}}{\overline{a}}, \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\overline{a}\overline{b}}{\underline{a}^2}\right)$, $\overline{x}_{\min} = \min\left(\frac{\overline{b}}{\underline{a}}, \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\underline{a}\underline{b}}{\overline{a}^2}\right)$.

Expressions (8) define all possible solutions of Eq. (4). The values of $\overline{x}_{min}$, $\underline{x}_{max}$ constitute the interval which produces the widest interval zero after substitution of them in Eq. (4). In other words, the maximum interval solution's width $w_{\max} = \overline{x}_{\min} - \underline{x}_{\max}$ corresponds to the maximum value of y: $y_{\max} = \frac{\overline{a}\overline{b}}{\underline{a}} - \underline{b}$. Substitution of degenerated solution $\underline{x} = \overline{x} = x_m$ in Eq. (4) produces the minimum value of y: $y_{min} = \frac{\overline{a}\cdot\overline{b} - \underline{a}\cdot\underline{b}}{\underline{a}+\overline{a}}$. It is clear that for any permissible solution $\underline{x}' > \underline{x}_{max}$ we have corresponding $\overline{x}' < \overline{x}_{min}$, for each $\underline{x}'' > \underline{x}'$ the inequalities $\overline{x}'' < \overline{x}'$ and $y'' < y'$ take place. Thus, the formal interval solution (8) factually represents the continuous set of nested interval solutions of Eq.(4). It is shown in [10] that this set of interval solutions can be naturally interpreted as a fuzzy number. We can see that values of y characterize the closeness of right hand side of Eq. (4) to degenerated zero and minimum value y_{min} is defined exclusively by interval parameters $[a]$ and $[b]$. Hence, the values of y may be considered, in a certain sense, as a measure of interval solution's uncertainty caused by the initial uncertainty of Eq. (4). Therefore we introduce

$$\alpha = 1 - \frac{y - y_{min}}{y_{max} - y_{min}}, \tag{9}$$

which may be treated as a certainty degree of interval solution of Eq. (4). We can see that α rises from 0 to 1 with decreasing of interval's width from maximum value to 0, i.e., with increasing of solution's certainty. Consequently, the values of α may be treated as labels of α-cuts representing some fuzzy solution of Eq. (4). Finally, the solution is obtained in form of triangular fuzzy number

$$\tilde{x} = \left\{\underline{x}_{\max}, \frac{\underline{b} + \overline{b}}{\underline{a} + \overline{a}}, \overline{x}_{\min}\right\}. \tag{10}$$

In a similar way, the fuzzy solutions of Eq. (4) were obtained for other placements of intervals $[a]$ and $[b]$ ([10,11]).

In the case of $[a] < 0$, $[b] > 0$, i.e., $\underline{a}, \overline{a} < 0$, $\underline{b}, \overline{b} > 0$ we get $\underline{a}\underline{x} - \overline{b} + \overline{a}\overline{x} - \overline{b} = 0$, $\tilde{x} = \left\{ \underline{x}_{\max}, \frac{\underline{b}+\overline{b}}{\underline{a}+\overline{a}}, \overline{x}_{\min} \right\}$, $\underline{x}_{\max} = \max(\frac{\overline{b}}{\overline{a}}, \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\overline{a}\underline{b}}{\underline{a}^2})$, $\overline{x}_{\min} = \min(\frac{\underline{b}}{\underline{a}}, \frac{\underline{b}+\overline{b}}{\overline{a}} - \frac{\underline{a}\overline{b}}{\overline{a}^2})$.

In the case of $[a] > 0$, $[b] < 0$, i.e., $\underline{a}, \overline{a} > 0$, $\underline{b}, \overline{b} < 0$ we get $\overline{a}\underline{x} - \overline{b} + \underline{a}\overline{x} - \underline{b} = 0$, $\tilde{x} = \left\{ \underline{x}_{\max}, \frac{\underline{b}+\overline{b}}{\underline{a}+\overline{a}}, \overline{x}_{\min} \right\}$, $\underline{x}_{\max} = \max\left(\frac{\underline{b}}{\underline{a}}, \frac{\underline{b}+\overline{b}}{\overline{a}} - \frac{\underline{a}\overline{b}}{\overline{a}^2}\right)$, $\overline{x}_{\min} = \min\left(\frac{\overline{b}}{\overline{a}}, \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\overline{a}\overline{b}}{\underline{a}^2}\right)$.

In the case of $[a] < 0$, $[b] < 0$, i.e., $\underline{a}, \overline{a} < 0$, $\underline{b}, \overline{b} < 0$ we get $\underline{a}\overline{x} - \overline{b} + \overline{a}\underline{x} - \underline{b} = 0$, $\tilde{x} = \left\{ \underline{x}_{\max}, \frac{\underline{b}+\overline{b}}{\underline{a}+\overline{a}}, \overline{x}_{\min} \right\}$, $\underline{x}_{\max} = \max\left(\frac{\overline{b}}{\underline{a}}, \frac{\underline{b}+\overline{b}}{\overline{a}} - \frac{\underline{a}\overline{b}}{\overline{a}^2}\right)$, $\overline{x}_{\min} = \min\left(\frac{\underline{b}}{\overline{a}}, \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\overline{a}\overline{b}}{\underline{a}^2}\right)$.

In the case of $[a] > 0, 0 \in [b]$, we get $\overline{a}\underline{x} - \overline{b} + \overline{a}\overline{x} - \underline{b} = 0$, $\tilde{x} = \left\{ \underline{x}_{\max}, \frac{\underline{b}+\overline{b}}{2\overline{a}}, \overline{x}_{\min} \right\}$, $\underline{x}_{\max} = \max\left(\frac{\underline{b}}{\underline{a}}, \frac{\underline{b}+\overline{b}}{\overline{a}} - \frac{\overline{b}}{\underline{a}}\right)$, $\overline{x}_{\min} = \min\left(\frac{\overline{b}}{\underline{a}}, \frac{\underline{b}+\overline{b}}{\overline{a}} - \frac{\underline{b}}{\underline{a}}\right)$.

In the case of $[a] < 0, 0 \in [b]$, we get $\underline{a}\overline{x} - \overline{b} + \underline{a}\underline{x} - \underline{b} = 0$, $\tilde{x} = \left\{ \underline{x}_{\max}, \frac{\underline{b}+\overline{b}}{2\overline{a}}, \overline{x}_{\min} \right\}$, $\underline{x}_{\max} = \max\left(\frac{\overline{b}}{\overline{a}}, \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\underline{b}}{\overline{a}}\right)$, $\overline{x}_{\min} = \min\left(\frac{\underline{b}}{\overline{a}} \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\overline{b}}{\overline{a}}\right)$.

Obviously, the support of obtained fuzzy number, i.e., the widest interval solution, can considered as a solution of Eq.(4) or the result of modified interval division. Hereinafter such result will be denoted as $[x]_{\text{ mod }}$. On the other hand, it seems natural to utilize all additional information available in the fuzzy solution.

The resulting fuzzy solution can be reduced to the interval one using well known defuzzification procedures. In our case, defuzzified left and right boundaries of the solution can be represented as follows:

$$\underline{x}_{def} = \frac{\int_0^1 \underline{x}(\alpha)d\alpha}{\int_0^1 d\alpha}, \overline{x}_{def} = \frac{\int_0^1 \overline{x}(\alpha)d\alpha}{\int_0^1 d\alpha} \tag{11}$$

For example, in the case of $[a], [b] > 0$, in [10,11] from (5), (9) and (11) the following expressions have been obtained:$\underline{x}_{def} = \frac{\overline{b}}{\underline{a}} - \frac{y_{max}+y_{min}}{2\underline{a}}, \overline{x}_{def} = \frac{\underline{b}}{\overline{a}} + \frac{y_{max}+y_{min}}{2\overline{a}}$.

Hereinafter, such interval solutions which can be treated also as the results of modified interval division will be denoted for all placements of $[a]$ and $[b]$ as $[x]_{\text{ mod } def}$.

It is shown in [10,11] that proposed method provides the considerable reducing of resulting interval's length in comparison with that obtained using conventional interval arithmetic rules.

3 The Use of Modified Interval Division for the Solution of Systems of Interval Linear Equations

The developed method is based on the modification of usual interval Gauss elimination procedure ($UIGEP$) which briefly can be presented as follows.

Let $[A][x] = [b]$ be a system of interval linear equations, where $[A]$ is $n \times n$ interval matrix with interval entries $[a_{ij}]$, $[b]$ is a column interval vector with n entries $[b_i]$, $[x]$ is an interval solution vector.

In the Forward Elimination stage, the system is reduced to the triangular form using elementary row operations:

$$[a_{ij}]^{(k+1)} = [a_{ij}]^{(k)} - \frac{[a_{ik}]^{(k)}[a_{kj}]^{(k)}}{[a_{kk}]^{(k)}},$$

$$[b_i]^{(k+1)} = [b_i]^{(k)} - \frac{[a_{ik}]^{(k)}[b_k]^{(k)}}{[a_{kk}]^{(k)}},$$

$$[a_{kk}]^{(k)} \neq 0, \qquad (k = 1, 2, ..., n-1; i, j = k+1, k+2, ..., n), \tag{12}$$

where k is the row number.

In the Backward Elimination stage, the interval solution vector is obtained as follows:

$$[x_n] = \frac{[b_n]^{(n)}}{[a_{nn}]^{(n)}}, [x_i] = \frac{[b_i]^{(i)} - \sum_{j=i+1}^{n} [a_{ij}]^{(i)}[x_j]}{[a_{ii}]^{(i)}},$$

$$(i = n-1, n-2, ..., 2, 1). \tag{13}$$

The usual interval arithmetic rules have been used in $UIGEP$. To improve numerical stability of the above algorithm, the Partial pivoting based on the means of interval entries has been employed in the Forward Elimination stage.

To get the modified interval Gauss elimination procedure ($MIGEP$), the usual interval division operation in (12) and (13) was substituted for the modified interval division presented in Section 2.

The two versions of $MIGEP$ were examined: the first is based on the widest interval result of modified division $[x]_{\text{mod}}$, the second - on the defuzzified result of modified division $[x]_{\text{mod } def}$ (see Section 2).

Therefore, the solutions obtained using $UIGEP$ and $MIGEP$ will be denoted as $[x_i]$, $[x_i]_{\text{mod}}$, $[x_i]_{\text{mod } def}$, $i = 1, ..., n$, respectively.

The seven well known from the literature examples were used to compare the methods:

Example 1 from [2]:

$$A = \begin{bmatrix} [0.7, 1.3] & [-0.3, 0.3] & [-0.3, 0.3] \\ [-0.3, 0.3] & [0.7, 1.3] & [-0.3, 0.3] \\ [-0.3, 0.3] & [-0.3, 0.3] & [0.7, 1.3] \end{bmatrix} \quad B = \begin{bmatrix} [-14, 7] \\ [9, 12] \\ [3, 3] \end{bmatrix}$$

Example 2 from [9]:

$$A = \begin{bmatrix} [3.7, 4.3] & [-1.5, -0.5] & [0.0, 0.0] \\ [-1.5, -0.5] & [3.7, 4.3] & [-1.5, -0.5] \\ [0.0, 0.0] & [-1.5, -0.5] & [3.7, 4.3] \end{bmatrix} \quad B = \begin{bmatrix} [-14, 14] \\ [9, 9] \\ [-3, 3] \end{bmatrix}$$

Example 3 from [9]:

$$A = \begin{bmatrix} [3.7, 4.3] & [-1.5, -0.5] & [0.0, 0.0] \\ [-1.5, -0.5] & [3.7, 4.3] & [-1.5, -0.5] \\ [0.0, 0.0] & [-1.5, -0.5] & [3.7, 4.3] \end{bmatrix} \quad B = \begin{bmatrix} [-14, 0] \\ [-9, 0] \\ [-3, 0] \end{bmatrix}$$

Example 4 from [9]:

$$A = \begin{bmatrix} [3.7, 4.3] & [-1.5, -0.5] & [0.0, 0.0] \\ [-1.5, -0.5] & [3.7, 4.3] & [-1.5, -0.5] \\ [0.0, 0.0] & [-1.5, -0.5] & [3.7, 4.3] \end{bmatrix} \quad B = \begin{bmatrix} [0, 14] \\ [0, 9] \\ [0, 3] \end{bmatrix}$$

Example 5 from [9]:

$$A = \begin{bmatrix} [3.7, 4.3] & [-1.5, -0.5] & [0.0, 0.0] \\ [-1.5, -0.5] & [3.7, 4.3] & [-1.5, -0.5] \\ [0.0, 0.0] & [-1.5, -0.5] & [3.7, 4.3] \end{bmatrix} \quad B = \begin{bmatrix} [2, 14] \\ [-9, -3] \\ [-3, 1] \end{bmatrix}$$

Example 6 from [9]:

$$A = \begin{bmatrix} [3.7, 4.3] & [-1.5, -0.5] & [0.0, 0.0] \\ [-1.5, -0.5] & [3.7, 4.3] & [-1.5, -0.5] \\ [0.0, 0.0] & [-1.5, -0.5] & [3.7, 4.3] \end{bmatrix} \quad B = \begin{bmatrix} [2, 14] \\ [3, 9] \\ [-3, 1] \end{bmatrix}$$

Example 7 from [2]:

$$A = \begin{bmatrix} [2, 3] & [0, 1] \\ [1, 2] & [2, 3] \end{bmatrix} \quad B = \begin{bmatrix} [0, 120] \\ [60, 240] \end{bmatrix}$$

The numbers of the examples correspond to the numbers in Table 1. These examples are characterized by some specific features and can be considered as critical tests. In the Table 1, the results obtained with use of usual interval Gauss elimination procedure ($[x_i]$), modified interval Gauss elimination procedure ($[x_i]_{\text{mod}}$ and $[x_i]_{\text{mod } def}$) are compared with those obtained using Markov's Jacobi type iteration method as the Markov's results can be treated as the maximal inner solution.

It is seen that only in the examples 2,5 and 7 the Markov's method provides non inverted interval solutions which can be treated as inner interval estimates of the solution set since it was proved by Shary [12] that "If a proper (non inverted) interval vector $[x]$ is a formal solution to the equation $[A][x] = [b]$ then $[x]$ is an inner interval estimate of the solution set." Hence, inverted solutions obtained by Markov in the other examples are not maximal inner solutions. Of course in the frameworks of directed interval arithmetic, "modular" arithmetic or the extended interval arithmetic developed by Kaucher, inverted interval solutions make a sense from purely mathematical point of view. But it is rather impossible to interpret inverted intervals in economic or mechanic terms. We can see that in the examples 2,5 and 7, the results obtained with use of our methods are close enough to the Markov's solutions (an exception is the example 7 where Markov's method provides the considerable narrower solution than our method). But the most important is the fact that in all considered examples, our methods provide non inverted interval solutions which are considerable more narrow that those obtained with use of usual interval Gauss Elimination procedure.

In [11], the proposed method has been tested on the examples of greater sizes of interval matrix $[A]$. To do this, three interval Leontiev's technological coefficients matrices 10×10, 100×100 and 500×500 were used. It was shown that obtained results are not charged by the considerable excess width effect and are substantially more narrow that those obtained with use of usual interval Gauss Elimination procedure.

Table 1. The comparison of obtained solutions

	Markov's method	$[x_i]_{\text{mod } def}$	$[x_i]_{\text{mod}}$ without deffuzification	$[x_i]$
1.	[-9.13, -13.05] [16.77, 7.16] [11, -2.68]	[-14.19, 8.80] [3.45, 11.60] [-8.59, 12.43]	[-37.29, 31.91] [-10.22, 25.21] [-22.12, 25.92]	[-101, 91] [-62.25, 99] [-66, 90]
2.	[-2.93, -2.93] [-0.94, -0.94] [-0.37, -0.37]	[-2.43, 2.43] [-2.67, 2.67] [-1.50, 1.50]	[-6.30, 6.30] [-6.21, 6.21] [-3.15, 3.15]	[-6.38, 6.38] [-6.40, 6.40] [-3.40, 3.40]
3.	[-3.46, -0.94] [-2.31, -1.77] [-0.90, -0.94]	[-3.40, -1.28] [-3.16, -1.19] [-1.62, -0.54]	[-4.73, 0.00] [-4.23, 0.00] [-2.25, 0.00]	[-6.38, 0.00] [-6.40, 0.00] [-3.40, 0.00]
4.	[0.94, 3.46] [1.77, 2.31] [0.94, 0.90]	[1.28, 3.40] [1.19, 3.16] [0.54, 1.62]	[0.00, 4.73] [0.00, 4.23] [0.00, 2.25]	[0.00, 6.38] [0.00, 6.40] [0.00, 3.40]
5.	[0.39, 2.87] [-1.11, -1.09] [-0.82, -0.18]	[0.67, 2.41] [-1.82, -0.07] [-1.17, 0.05]	[-0.69, 3.76] [-3.03, 1.17] [-1.84, 0.71]	[-1.09, 4.29] [-4.02, 1.24] [-2.44, 0.78]
6.	[1.46, 3.54] [2.46, 2.28] [0.11, 0.52]	[1.69, 3.62] [1.66, 3.32] [-0.15, 1.23]	[0.52, 4.82] [0.46, 4.27] [-0.87, 2.02]	[0.52, 6.25] [0.45, 6.07] [-0.88, 2.73]
7.	[0, 17.14] [30, 68.57]	[-12.4, 30.33] [-1.33, 67.55]	[-53.22, 71.74] [-42.00, 106.44]	[-120, 90] [-60, 240]

4 Conclusion

The aim of the paper is to present a new modified approach to interval division based on the concept of "interval extended zero" method [10,11] and its application to the solution of the systems of interval linear equations. To illustrate a new method, we present the results obtained for known seven examples repeatedly used in the literature as the tests for numerical methods in the interval setting. Comparing our results with those obtained using Markov's Jacobi type iterative method and usual interval Gauss elimination procedure, we show that the proposed method not only allows us to decrease the excess width effect, but makes it possible to avoid inverted interval solutions too. It is important that our results are close to the so-called maximal inner solutions, i.e., approximate solutions with minimal excess width effect.

References

1. Cleary, J.C.: Logical Arithmetic. Future Computing Systems 2, 125–149 (1987)
2. Hansen, E.: Bounding the solution of interval linear equations. SIAM J. Numer. Anal. 29(5), 1493–1503 (1992)
3. Kearfott, B.: Rigorous Global Search: Continuous Problems. Kluwer Academic Publishers, The Netherlands (1996)

4. Kupriyanova, L.: Inner estimation of the united solution set to interval linear algebraic system. Reliable Computing 1(1), 15–31 (1995)
5. Leontief, W.: Quantitative input-output relations in the economic system of the United States. Review of Economics and Statistics 18, 100–125 (1936)
6. Markov, S.: An iterative method for algebraic solution to interval equations. Applied Numerical Mathematics 30, 225–239 (1999)
7. Markov, S., Popova, E., Ullrich, C.: On the Solution of Linear Algebraic Equations Involving Interval Coefficients. In: Margenov, S., Vassilevski, P. (eds.) Iterative Methods in Linear Algebra II, IMACS Series in Computational and Applied Mathematics, vol. 3, pp. 216–225 (1996)
8. Moore, R.E.: Interval analysis. Prentice-Hall, Englewood Cliffs (1966)
9. Ning, S., Kearfott, R.B.: A comparison of some methods for solving linear interval equations. SIAM J. Numer. Anal. 34(4), 1289–1305 (1997)
10. Sevastjanov, P., Dymova, L.: Fuzzy solution of interval linear equations. In: Proc. of 7th Int. Conf. Paralel Processing and Applied Mathematics, Gdansk, pp. 1392–1399 (2007)
11. Sevastjanov, P., Dymova, L.: A new method for solving interval and fuzzy equations: linear case. Information Sciences 17, 925–937 (2009)
12. Shary, S.P.: A New Technique in Systems Analysis under Interval Uncertainty and Ambiguity. Reliable Computing 8, 321–418 (2002)
13. Zyuzin, V.: An iterative method for solving system of algebraic segment equations of the first order. In: Differential Equations and the Theory of Functions, pp. 72–82. Saratov State University, Saratov (1989) (in Russian)

Remarks on Algorithms Implemented in Some C++ Libraries for Floating-Point Conversions and Interval Arithmetic

Malgorzata A. Jankowska

Poznan University of Technology, Institute of Applied Mechanics
Piotrowo 3, 60-965 Poznan, Poland
malgorzata.jankowska@put.poznan.pl

Abstract. The main aim of the paper is to give a short presentation of selected conversion functions developed by the author. They are included in two C++ libraries. The *FloatingPointConversion* library is dedicated for conversions in the area of floating-point numbers and the second one, the *IntervalArithmetic* library, carries out the similar task for interval values as well as supports computations in the floating-point interval arithmetic with a suitable *CInterval* class. The functions considered are all intended to be used with the Intel Architectures (i.e. the IA-32 and the IA-64) and dedicated for C++ compilers that specify 10 bytes for the *long double* data type.

Keywords: IEEE standard 754 for binary floating-point arithmetic, floating-point interval arithmetic, floating-point conversions.

1 Introduction

The concept of floating-point interval arithmetic and reliable computing was a natural consequence of a desire of scientists and engineers to develop such methods and tools that would ensure some kind of accuracy of results obtained. The development of interval methods and interval arithmetic entailed the appearance of many different libraries and even computational environments that implement floating-point interval arithmetic in computer. Let us mention at least some of them starting from the *XSC* (*eXtensions for Scientific Computation*) languages, which include *Pascal-XSC* and *C-XSC* (see e.g. [3]). Then, the *COSY* that is a language for verified global optimization and ODE solving and is based on intervals and Taylor models with reminder bounds (see e.g. [14]). Next the *VNODE* which is a C++ package for computing rigorous bounds on the solution of the IVP for ODEs (see e.g. [18]). Furthermore, there are the C++ and Fortran 95 interval arithmetic libraries provided with the Sun Studio C++ compiler and the *IntBLAS* that is a C++ Interval BLAS (*Basic Linear Algebra Subprograms*) implementation (for more information on the different libraries and computational environments see also *Languages for Interval Computations* at http://www.cs.utep.edu/interval-comp/).

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 436–445, 2010.

Regardless of that Marciniak proposed the *IntervalArithmetic* unit (for the last version 2.13 see [15]) which was originally written in Delphi Pascal. Shortly afterwards, Jankowska translated the code to Borland C++ language. The *IntervalArithmetic* unit supports all the functionality needed for computations on intervals. Hence, it became a part of the *OOIRK* (*Object Oriented Interval Runge-Kutta*) system which is dedicated for solving the IVP for ODEs by interval methods of Runge-Kutta type (see e.g. [15]). Similarly, its Borland C++ version 1.0 was used for the development of the *IMMSystem* (*Interval Multistep Methods System*) designed for solving the IVP for ODEs by the interval multistep methods of Adams type (see e.g. [13]).

The plans of the author to perform parallel computations on multi-core processors influenced the decision about changing a compiler to the Intel C++ compiler for Windows (or Linux). It occurred soon that the direct transfer of the library code and recompilation with the Intel C++ compiler was impossible. Some problems arose because there is no support for the *printf* function for 80-bit *long double* values in any Intel or Microsoft Windows standard libraries. We can use the *long double* data type that is 80-bit floating-point (with */Qlong-double* compiler option) but before we print a number we have to convert it to *double*. In this way we cannot see all decimal digits of significand which are available due to the 80-bit floating-point format. Furthermore, even though the detailed tests of standard conversion functions were not in the area of my interest it occurred soon that there are also some problems with the *_atoldbl* member function of the Microsoft C++ *stdlib.h* library. This function converts a *string* that represents a real number to a *long double* value, but as one can check (see e.g. [7]) we have as follows:

- the conversion of a *string* to a *long double* value fails for some denormalized numbers,
- the return value sometimes gives misleading information about the underflow exception,
- the conversion process succeeds even if a string value does not represent a real number correctly.

All these facts influenced the decision to develop the *FloatingPointConversion* library (for the last version 1.1 see [7]) and a new set of functions for the *IntervalArithmetic* library (the last version 2.1 is given in [8]).

The C++ libraries considered are dedicated for:

- the Intel Architecture (the IA-32 and the IA-64) that defines three floating-point data types: single-precision, double-precision and double extended-precision (note that the data formats for these data types are compliant with the IEEE Standard 754-1985 (see [4])),
- the C++ compilers that specify 10 bytes for the *long double* data type to store the floating-point numbers (for detailed information on the implementation of the C++ floating-point data types i.e. *float*, *double* and *long double* in accordance to selected 16-, 32- and 64-bit C++ compilers see e.g. [1]).

Up to now the libraries have been carefully tested with the Intel C++ compiler 11.0.074 [IA-32] for Windows. As soon as possible, they will be moved to Linux operating system and then tested with Intel C++ compiler for Linux and GCC compiler.

Algorithms of conversion functions proposed in the next sections are based on the rules set in the IEEE Standard 754 for Binary Floating-Point Arithmetic (see e.g. [6], [19]) about the real numbers representation in computer. Let us only mention that the processor represents real numbers in the following form:

$$(-1)^s \, 2^E \, (b_{0\Delta} b_1 b_2 \ldots b_{p-1}) \,, \tag{1}$$

where $s = 0$ or 1 is the sign field that states if the number is positive or negative, E is a decimal value of exponent, the significand field is represented by $b_{0\Delta} b_1 b_2 \ldots b_{p-1}$, where $b_i = 0$ or 1, p is the number of precision bits, and $_\Delta$ indicates an assumed binary point. The real numbers are stored in a three-field binary format:

"sign exponent (in biased form) significand (i.e. integer and fraction)",

where the biased exponent equals the actual exponent E increased by a constant (also called an exponent bias) which is dependent on the number of bits used to store the biased exponent and hence, it is different for each data type. Some information on floating-point encodings for signed zeros, denormalized finite numbers, normalized finite numbers and signed infinities are also given in Table 1 (for details see [6], [19]).

Table 1. Floating-point encodings for signed zeros, denormalized finite numbers, normalized finite numbers and signed infinities

Class		Sign	Biased Exponent	Significand	
				J-bit	Fraction
Positive	$+\infty$	0	11…11	1	00…00
	+ Normals	0 . 0	11…10 … 00…01	1 . 1	11…11 … 00…00
	+ Denormals	0 . 0	00…00 … 00…00	0 . 0	11…11 … 00…01
	+ Zero	0	00…00	0	00…00
Negative	– Zero	1	00…00	0	00…00
	– Denormals	1 . 1	00…00 … 00…00	0 . 0	00…01 … 11…11
	– Normals	1 . 1	00…01 … 11…10	1 . 1	00…00 … 11…11
	$-\infty$	1	11…11	1	00…00
Data type	single precision:		← 8 bits →		← 23 bits →
	double precision:		← 11 bits →		← 52 bits →
	double extended precision:		← 15 bits →		← 63 bits →

In the paper we present just a short description of the functions designed for the floating-point conversions (see Section 2) as well as for the conversions in the area of the floating-point intervals (see Section 3). Additionally, two algorithms for the conversion of a *string* representing a real number given in the

decimal format to a *long double* floating-point number (i.e. the *ExStrTo_LDouble* function) and a *long double* floating-point number to a *string* that represents it in the decimal exponent format (i.e. the *LDoubleTo_ExponentStr* function) are proposed. Finally, we give some conclusions and formulate future plans.

2 C++ Library for Floating-Point Conversions

All functions defined in the *FloatingPointConversion* library can be divided into two groups in accordance to a kind of conversion. Hence, we can convert:

- a *string* that represents a real number given in the decimal format (the *StrTo_LDouble* and *ExStrTo_LDouble* functions) or in the binary double extended-precision format (the *BinaryStrTo_LDouble* function) to a *long double* floating-point number,
- a *long double* floating-point number to a *string* that represents a given number in the binary double extended-precision format (the *LDoubleTo_BinaryStr* function), the decimal fixed-point format (the *LDoubleTo_FixedPointStr* function) and the decimal exponent format, also called the scientific format (the *LDoubleTo_ExponentStr* function).

Most of these functions make use of some enumerated data types defined in the unit and given in the following list:

```
enum TRoundingMode {rmNearest,rmDown,rmUp,rmZero};
enum TFloatingPointNumber {fpnPositiveZero,fpnNegativeZero,
     fpnDenormal,fpnNormal,fpnPosInfinity,fpnNegInfinity};
enum TException {exNone,exUnderflow,exOverflow,exConversionError};
enum TFloatingPointConstant {fpcZero,fpcPositiveZero,fpcNegativeZero,
     fpcSmallestPositiveDenormal,fpcLargestPositiveDenormal,
     fpcSmallestPositiveNormal,fpcLagestPositiveNormal,
     fpcSmallestNegativeDenormal,fpcLargestNegativeDenormal,
     fpcSmallestNegativeNormal,fpcLagestNegativeNormal,
     fpcPositiveInfinity,fpcNegativeInfinity};
```

Now let us propose the algorithm of the *ExStrTo_LDouble* function. It is based only on standard C++ language features and hence, it is independent of the implementation of C++ language and standard conversion functions, provided that such C++ implementation meets ISO/IEC 14882:2003 standard.

```
Function:
  long double ExStrTo_LDouble (const string         & sx,
                               TException           & exception,
                               bool                 & exact_result,
                               TFloatingPointNumber & floating_point_number,
                               TRoundingMode          rounding = rmNearest);
Input parameters:
  sx - a string that is converted into a long double floating-point number;
  rounding (a default value is rmNearest) - a rounding mode; it is used if the number
      given by the string sx cannot be represented exactly in the double extended-precision
      format;
```

```
Output parameters:
  exception - informs if any of possible exceptions occurred during the conversion process;
      if so, it gives a kind of exception (see TException data type);
  exact_result - is equal to true if the number given in the string sx has the exact
      representation in the double extended-precision format; otherwise its value is equal
      to false;
  floating_point_number - gives the information about the class, that the floating-point
      number converted from a string sx, belongs to;

Return value:
  It is equal to a floating-point number which is an exact representation of a real number
  given in the string sx, or if such exact conversion is not possible, it is equal to
  a floating-point number chosen according to the rounding mode.

Utility functions:
  long double GetLDouble (char sign, vector<char> bnumber, int exponent)
      - a function returns a floating-point number which is the exact representation
      in the double extended-precision format of a real number given in the string sx;

  long double GetLDouble_Rounded (char sign, vector<char> bnumber, int exponent,
                                  TRoundingMode rounding, TException & exception,
                                  TFloatingPointNumber & floating_point_number)
      - a function that is called when a real number given in the string sx does not have
      the exact representation in the double extended-precision format; a return value is
      then a floating-point number chosen according to the rounding mode specified in the
      rounding parameter; furthermore, the exception and floating_point_number parameters
      are set;

  long double GetLDouble_Overflow (char sign, TRoundingMode rounding,
                                   TFloatingPointNumber & floating_point_number)
      - a function that is called when a real number given in the string sx is larger or
      equal to a floating-point number that represents the positive/negative infinity in
      the double extended-precision format; a return value is then a floating-point
      representation of the positive/negative infinity or the largest positive/negative
      normal number in accordance to the rounding mode specified in the rounding parameter;
      furthermore, the floating_point_number parameter is set;

  long double GetPositiveInfinity () - a function returns a floating-point number that
      is equal to the positive infinity in the double extended-precision format;

  long double GetNegativeInfinity () - a function returns a floating-point number that
      is equal to the negative infinity in the double extended-precision format;

Remarks:
  a size of a given string or a vector object is the number of its elements;

Algorithm:
Step 1: sa = sx

Step 2: Convert a string value sa into the form that shows its decimal fixed-point repre-
        sentation as follows:
            sign(+/-) decimal_integer_part separator(,/.) decimal_fraction_part

Step 3: sign = '+', if a given number is positive;
        otherwise, sign = '-'

Step 4: is_frac_part_zero = false, if there is a nonzero decimal fraction part
        otherwise, is_frac_part_zero = true

Step 5: if (sa == +0,0) or (sa == +0.0), then
            exception = exNone
            exact_result = true
            floating_point_number = fpnPositiveZero
            return +0.0
```

```
Step 6: if (sa == -0,0) or (sa == -0.0), then
            exception = exNone
            exact_result = true
            floating_point_number = fpnNegativeZero
            return -0.0

Step 7: Find the binary representation of an integer part of a real number given in sa and
        save it in the bnumber vector object. Note that only first 16386 binary digits
        are required.

Step 8: exponent = the size of bnumber - 1

Step 9: comment: Check if the overflow exception occurred ...
        if (exponent == 16384) or (exponent == 16385), then

            comment: Check if there is the exact representation of the infinity ...
            if (exponent == 16384) and (is_frac_part_zero == true), then

                if (bnumber[1] == 0, ... , bnumber[16384] == 0), then
                    exception = exOverflow
                    exact_result = true

                    if (sign == '+'), then
                        floating_point_number = fpnPositiveInfinity
                        return GetPositiveInfinity()
                    else
                        floating_point_number = fpnNegativeInfinity
                        return GetNegativeInfinity()

            comment: The true result has to be rounded ...
            exception = exOverflow
            exact_result = false
            return GetLDouble_Overflow(sign,rounding,floating_point_number)

Step 10: comment: All 64 precision bits are used. Hence, if there is no fraction part
         and exponent >= 63, then a given number has the exact representation or can be
         rounded according to the current rounding mode. Similarly, if exponent >= 66.
         Note that if it is possible we take no less than three additional precision bits
         to round the true result to the nearest floating-point value correctly.

         if ((exponent >= 63) and (is_frac_part_zero == true)) or (exponent >= 66), then

             if (bnumber[64] == 0, ... , bnumber[exponent] == 0), then
                is_nonzero_digit = false
             else
                is_nonzero_digit = true

             comment: Check if there is the exact representation of a given number ...
             if (is_frac_part_zero == true) and (is_nonzero_digit == false)
                 exception = exNone
                 exact_result = true
                 floating_point_number = fpnNormal
                 return GetLDouble(sign,bnumber,exponent)

             comment: The true result has to be rounded ...
             else
                 exact_result = false
                 return GetLDouble_Rounded(sign,bnumber,exponent,rounding,exception,
                                           floating_point_number)

Step 11: comment: Check if there is a nonzero integer part of a given number ...
         if (bnumber[0] == 1), then

             Find the binary representation of a fraction part of a given number and
             add it to the bnumber vector object. Note that only first 67 binary digits
             in bnumber are required.
```

```
            comment: Check if there is the exact representation of a given number ...
            if (size of bnumber - 1 <= 63)
                exception = exNone
                exact_result = true
                floating_point_number = fpnNormal
                return GetLDouble(sign,bnumber,exponent)

            comment: The true result has to be rounded ...
            else
                exact_result = false
                return GetLDouble_Rounded(sign,bnumber,exponent,rounding,exception,
                                          floating_point_number)

Step 12: comment: A given number is a decimal fraction ...

        Find the normalized binary representation of a fraction part of a given number
        and save it in the bnumber vector object with exponent <= 16382.

        If the exponent value reaches 16383, then find the appropriate denormalized
        representation. Note that exponent is an absolute value of the true exponent.
        Furthermore, only first 67 binary digits in bnumber are required.

        if (exponent <= 16382)

            if (size of bnumber - 1 <= 63)
                exception             = exNone
                exact_result          = true
                floating_point_number = fpnNormal
                return GetLDouble(sign,bnumber,-1*exponent)
           else
               exact_result = false
               return GetLDouble_Rounded(sign,bnumber,-1*exponent,rounding,exception,
                                         floating_point_number)

        else if (exponent == 16383)

            if (size of bnumber - 1 <= 63)
                exception             = exUnderflow
                exact_result          = true
                floating_point_number = fpnDenormal
                return GetLDouble(sign,bnumber,-1*exponent)
           else
               exact_result = false
               return GetLDouble_Rounded(sign,bnumber,-1*exponent,rounding,exception,
                                         floating_point_number)

Step 13:
    exception = exConversionError
    return +0.0
```

Probably, the most widely used function is *LDoubleTo_ExponentStr*. Hence, let us now shortly explain the way it works (see also [7]).

```
Function:
  string LDoubleTo_ExponentStr (long double x,
                                short precision = 18,
                                short digits = 4,
                                TRoundingMode rounding = rmNearest);

Input parameters:
  x - a long double floating-point number to be converted into a string;
  precision (by default it is equal to 18) - a parameter that sets the number of decimal
      significant digits with one digit before and precision - 1 digits after the decimal
      separator; it can be any integer that satisfies the condition 2 ≤ precision ≤ 18;
```

```
  digits (by default it is equal to 4) - a parameter that sets the minimal number
      of decimal exponent digits; it can be any integer that satisfies the condition
      1 ≤ digits ≤ 4;
  rounding (a default value is rmNearest) - a parameter that sets the rounding mode which
      is used when the floating-point number x cannot be represented exactly with a given
      precision;

Return value:
  It is equal to a string value that gives the representation of x in the decimal
  scientific format, i.e. the number is shown in the following form:
     ''sign (+/-)  significand (i.e. integer and fraction) (precision decimal digits)
                 E(+/-) exponent (at least digits decimal digits)''.

Utility functions:
  string LDoubleTo_FixedPointStr (long double x)
      - a function that returns a string that gives the exact representation of x
      in the decimal fixed-point format, i.e. the number is shown in the following form:
         ''sign(+/)  decimal_integer_part  separator(, /.)  decimal_fraction_part'';

Local variables:
  sx - a string value that stores the return representation of x in the decimal scientific
       format;

Algorithm:
Step 1: sx = LDoubleTo_FixedPointStr(x)

Step 2: exponent ← the decimal exponent of the number represented by the string sx

Step 3: sx ← the significand of the number represented by the string sx; the maximum
        number of significant digits that are taken for rounding procedures equals 19 (with
        one digit before and 18 after the decimal separator; note the last digit is used
        only to make the rounding correctly); the remaining ones are neglected;

Step 4: switch (rounding)
          case rmZero:
            comment: Round the number represented by the string sx toward zero (chop) with
            the precision decimal digits and store the significand in sx (see [7]).
            DecimalRoundZero(sx,precision)
            break

          case rmNearest:
            comment: Round the number represented by the string sx to the nearest/even with
            the precision decimal digits and store the significand in sx;
            if it is required modify the value of the exponent parameter (see [7]).
            DecimalRoundNearest(sx,exponent,precision)
            break

          case rmDown:
            comment: Round the number represented by the string sx down with
            the precision decimal digits and store the significand in sx;
            if it is required modify the value of the exponent parameter (see [7]).
            DecimalRoundDown(sx,exponent,precision)
            break

          case rmUp:
            comment: Round the number represented by the string sx up with
            the precision decimal digits and store the significand in sx;
            if it is required modify the value of the exponent parameter (see [7]).
            DecimalRoundUp(sx,exponent,precision)
            break

Step 5: comment: Add the string representing the decimal exponent given in the exponent
        parameter to the significand stored in the string sx
        AddExponent(sx,exponent,digits)

Step 6: return sx
```

3 C++ Library for Interval Floating-Point Arithmetic

Now let us shortly present the new version of the *IntervalArithmetic* library. Taking advantage of some C++ language features we define the *CInterval* class in which the basic operators (i.e. addition, subtraction, multiplication and division) are overloaded. Hence, you can perform the elementary arithmetic operations on intervals as objects of the *CInterval* class in a reasonably simple way. Furthermore, there are also several useful conversion functions defined in the library. They carry out two main tasks. First one is to convert a string representing a real number given in the decimal format (the *StrTo_Interval* and *ExStrTo_Interval* functions) or in the binary double extended-precision format (the *BinaryStrTo_Interval* function) to a machine interval. We define a machine interval as an interval whose endpoints are equal or are two subsequent floating-point numbers. Such interval contains the real number represented by a given string. The *StrTo_Interval* function is similar to the one given by Marciniak (see e.g. [15]). Now in the *ExStrTo_Interval* function we propose another algorithm that is based on the similar idea as in the case of the *ExStrTo_LDouble* function (see Section 2). The second task (see the *IntervalTo_ExponentStr* function in [7]) is to convert a machine interval stored in the *CInterval* object to strings that represent its left and right endpoints in the decimal scientific format. The algorithm of the *IntervalTo_ExponentStr* function is originally based on the idea of Marciniak (see e.g. [15]). Its new version uses the *LDoubleTo_ExponentStr* function to get the appropriate string values of the endpoints and hence, we can also specify the number of digits of the significand and the exponent as well. In this way we control the decimal scientific format of the endpoints.

4 Conclusions

The author proposes the *FloatingPointConversion* library and the extended version of the *IntervalArithmetic* library. Furthermore, two selected algorithms for the floating-point conversions are described in detail (see Section 2). The libraries considered are dedicated for programmers dealing with scientific computations in conventional floating-point arithmetic as well as in floating-point interval arithmetic. The conversion functions and the *CInterval* class proposed are of great importance because most of them are frequently used especially when developing computational applications. As it occurred, they are not always supported in a given C++ compiler implementation or even if they are they can differ in name or the way they work. The usage of the libraries proposed makes the programmer independent of such problems. It was not the aim of the paper to present all the functions supported by the libraries. The user and reference guides for the libraries considered can be found in *Software* at http://www.mjank.user.icpnet.pl/. Furthermore, the appropriate static libraries are available from the author upon request. Up to now the static libraries have been created and carefully tested with the Intel C++ compiler 11.0.074 [IA-32] for Windows. They will be also moved to Linux operating system and then tested with Intel C++ compiler for Linux and GCC compiler.

References

1. Fog, A.: Calling conventions for different C++ compilers and operating systems, Copenhagen University College of Engineering (last updated in 2008), http://www.agner.org/optimize/
2. Gajda, K., Jankowska, M., Marciniak, A., Szyszka, B.: A Survey of Interval Runge-Kutta and Multistep Methods for Solving the Initial Value Problem. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 1361–1371. Springer, Heidelberg (2008)
3. Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: Numerical Toolbox for Verified Computing I. Basic Numerical Problems. Springer, Berlin (1993)
4. IEEE Computer Society, IEEE Standard 754 for Floating-Point Arithmetic (1985)
5. IEEE Computer Society, IEEE Standard 754 for Floating-Point Arithmetic (2008)
6. Intel 64 and IA-32 Architectures Software Developers Manual, Volume 1: Basic Architecture (2008)
7. Jankowska, M.A.: C++ Library for Floating-Point Conversions. User and Reference Guide, Poznan University of Technology. Last updated (2009), Software at http://www.mjank.user.icpnet.pl/
8. Jankowska, M.A.: C++ Library for Floating-Point Interval Arithmetic. User and Reference Guide, Poznan University of Technology (last updated 2009), Software at http://www.mjank.user.icpnet.pl/
9. Jankowska, M., Marciniak, A.: Implicit Interval Multistep Methods for Solving the Initial Value Problem. Computational Methods in Science and Technology 8(1), 17–30 (2002)
10. Jankowska, M., Marciniak, A.: On Explicit Interval Methods of Adams-Bashforth Type. Computational Methods in Science and Technology 8(2), 46–57 (2002)
11. Jankowska, M., Marciniak, A.: An Interval Version of the Backward Differentiation (BDF) Method. In: SCAN 2006 Conference Post-Proceedings IEEE-CPS Product No. E2821 (2007)
12. Jankowska, M., Marciniak, A.: On Two Families of Implicit Interval Methods of Adams-Moulton Type. Computational Methods in Science and Technology 12(2), 109–113 (2006)
13. Jankowska, M.: Interval Multistep Methods of Adams type and their Implementation in the C++ Language. Ph.D. Thesis, Poznan University of Technology, Poznan (2006)
14. Makino, K., Berz, M.: COSY INFINITY Version 9. Nuclear Instruments and Methods A558, 346–350 (2005)
15. Marciniak, A.: Selected Interval Methods for Solving the Initial Value Problem. Publishing House of Poznan University of Technology, Poznan (2009)
16. Jaulin, L., Kieffer, M., Didrit, O., Walter, É.: Applied Interval Analysis. Springer, London (2001)
17. Moore, R.E.: Interval Analysis. Prentice-Hall, Englewood Cliffs (1966)
18. Nedialkov, N.S., Jackson, K.R.: The Design and Implementation of a Validated Object-Oriented Solver for IVPs for ODEs, Technical Report 6, Software Quality Research Laboratory, Department of Computing and Software, McMaster University, Harnilton (2002)
19. Pentium Processor Family Developers Manual, Volume 3: Architecture and Programming Manual (1995)

An Interval Method for Seeking the Nash Equilibria of Non-cooperative Games

Bartłomiej Jacek Kubica and Adam Woźniak

Institute of Control and Computation Engineering, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland

Abstract. Computing Nash equilibria in continuous games is a difficult problem. In contrast to discrete games, algorithms developed for continues ones are rather inefficient. This paper proposes a new approach – making use of interval methods we try to solve the problem directly, seeking points that fulfill Nash conditions. We also consider a shared-memory parallelization of the proposed algorithm. Preliminary numerical results are presented. Some new practical aspects of interval methods are considered.

1 Introduction

A Nash equilibrium [12], defined first in 1950, is one of basic concepts in the game theory. For a non-cooperative game the Nash equilibrium is a situation (an assignment of strategies to all players), when it is not beneficial to any of the players to change their strategy unless others will do so.

Formally, let us consider a game with n players, each of them trying to choose the decision variable $x_i \in X_i$ to minimize the cost $q_i(x_1, \ldots, x_i, \ldots, x_n)$, $i = 1, \ldots, n$.

The tuple $(x_1^*, \ldots, x_n^*)$ is a Nash equilibrium, iff:

$$\forall i = 1, \ldots, n \quad \forall x_i \; q_i(x_1^*, \ldots, x_{i-1}^*, x_i, x_{i+1}^*, \ldots, x_n^*) \geq q_i(x_1^*, \ldots, x_n^*) \; . \tag{1}$$

In the simplest case the player's strategy is simply a tuple of numbers (vector) they choose from the given set, i.e. $x_i = \left(x_1^{(i)}, \ldots, x_{k_i}^{(i)}\right) \in X_i \subseteq \mathbb{R}^{k_i}$. Let us denote $x = (x_1, \ldots, x_n)$.

The structures of domains X_i may vary widely, defining different types of games. Most often various types of *discrete* games are considered (e. g. [15]).

However, continuous games are also an interesting and widely applicable model, encountered in several branches of economical sciences and decision making theory. Example applications include water resource systems [7], [16], so-called mechanism implementation in economy [18] and cellular networks [10].

Cellular power control game. There is a fixed number of agents – users of cellular transmitters in a cell. Each of them chooses the power level $p_i \in [0, p_{\max}]$. And each of them gets a level of SINR (signal-to-interference-and-noise-ratio), depending on his own transmission power level and levels of other users. When

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 446–455, 2010.

assuming a CDMA technique, a (minimized) cost of each agent can be computed, using the BER (bit-error-rate) function as:

$$u_i(p_1,\ldots,p_n) = \frac{R}{p_i}\cdot\Big(1-2BER\big(f(p_1,\ldots,p_n)\big)\Big)^L \ ,$$

where R is the transmission rate, L is the size of a packet (in bits) and $f(p_1,\ldots,p_n)$ is the SINR.

Computing Nash equilibria for a continuous game is in general much more difficult than for a discrete one. It is also less often considered in the literature. Most approaches to solve it belong to one of three classes:

- minimization of the function defined by Nikaidô and Isoda [14] or a similar one,
- Rosen type algorithms [17] for finding fixed points of a properly defined mapping,
- Gabay and Moulin [1] approach using differential calculus.

We propose an alternative method – making use of interval methods we try to solve the problem directly, seeking points that fulfill Nash conditions.

2 Basics of Interval Computations

Now, we shall define some basic notions of intervals and their arithmetic. We follow a widely acknowledged standards (cf. e.g. [2], [4], [5], [13]).

We define the (closed) interval $[\underline{x},\overline{x}]$ as a set $\{x\in\mathbb{R}\mid \underline{x}\le x\le\overline{x}\}$.

Following [6], we use boldface lowercase letters to denote interval variables, e.g. $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$, and $\mathbb{IR}$ denotes the set of all real intervals.

We design arithmetic operations on intervals so that the following condition was fulfilled: if we have $\odot\in\{+,-,\cdot,/\}$, $a\in\boldsymbol{a}$, $b\in\boldsymbol{b}$, then $a\odot b\in\boldsymbol{a}\odot\boldsymbol{b}$. The actual formulae for arithmetic operations (see e.g. [2], [4], [5]) are as follows:

$$\begin{aligned}
&[\underline{a},\overline{a}]+[\underline{b},\overline{b}] = [\underline{a}+\underline{b},\overline{a}+\overline{b}] \ ,\\
&[\underline{a},\overline{a}]-[\underline{b},\overline{b}] = [\underline{a}-\overline{b},\overline{a}-\underline{b}] \ ,\\
&[\underline{a},\overline{a}]\ \cdot\ [\underline{b},\overline{b}] = [\min(\underline{ab},\underline{a}\overline{b},\overline{a}\underline{b},\overline{a}\overline{b}),\max(\underline{ab},\underline{a}\overline{b},\overline{a}\underline{b},\overline{a}\overline{b})] \ ,\\
&[\underline{a},\overline{a}]\ /\ [\underline{b},\overline{b}] = [\underline{a},\overline{a}]\cdot\big[1\,/\,\overline{b},1\,/\,\underline{b}\big] \ , \qquad 0\notin[\underline{b},\overline{b}] \ .
\end{aligned}$$

The definition of interval vector $\boldsymbol{x}$, a subset of $\mathbb{IR}^n$ is straightforward: $\mathbb{IR}^n\supset\boldsymbol{x}=\boldsymbol{x}_1\times\cdots\times\boldsymbol{x}_n$. Traditionally interval vectors are called *boxes*.

Links between real and interval functions are set by the notion of an *inclusion function*, see e.g. [4]; also called an *interval extension*, e.g. [5].

Definition 1. *A function* $\mathsf{f}\colon\mathbb{IR}\to\mathbb{IR}$ *is an* inclusion function *of* $f\colon\mathbb{R}\to\mathbb{R}$, *if for every interval* $\boldsymbol{x}$ *within the domain of* f *the following condition is satisfied:*

$$\{f(x)\mid x\in\boldsymbol{x}\}\subseteq\mathsf{f}(\boldsymbol{x}) \ .$$

The definition is analogous for functions $f\colon\mathbb{R}^n\to\mathbb{R}^m$.

When computing interval operations, we can round the lower bound downward and the upper bound upward. This will result in an interval that will be a bit overestimated, but will be *guaranteed to contain the true result of the real-number operation.*

3 The Proposed Approach

The general schema is going to be a specific variant of the branch-and-bound (b&b) method (see e.g. [2], [5], [13]). The boxes are stored in a queue – a stack would be appropriate, too, but the queue is better for parallelization (see Section 4).

Thanks to the virtues of interval methods we may solve the problem more directly than by Nikaidô-Isoda type transformations. We shall seek for points satisfying the given conditions defined by (1).

Please note that these conditions say simply that *some functions should have minima in certain points.* So, from well-known optimality conditions (and assuming proper smoothness) we can derive necessary conditions for Nash equilibria. For unconstrained problems, it will be:

$$\frac{\partial q_i}{\partial x_j^{(i)}}(x) = 0 \quad i = 1, \ldots, n, \quad x_j \in \{x_1^{(i)}, \ldots, x_{k_i}^{(i)}\} \ . \tag{2}$$

The above equations form a system of N equations in N variables, where $N = \sum_{i=1}^{n} k_i$.

Handling arbitrary-constrained problems is a bit more difficult, but for unconstrained and bound-constrained problems, we can use the above conditions as a powerful rejection/reduction tool for the b&b algorithm. Obviously, the test cannot be used on boxes tangent to the bound constraints.

Similarly, we can get an analog of the monotonicity test ([2], [5], [13]) – checking if a function $q_i(\cdot)$ is monotonous with respect to any of the variables from V_i (monotonicity wrt other variables does not preclude existence of a Nash equilibrium!).

The b&b process terminates with the list of boxes that *possibly* contain Nash equilibria. Do they really do? To find out we should compare actual criteria values in these points and in proper subsets of the domain. It can be done in a few ways.

First of all, we can actually solve related optimization problems or constraint satisfaction problems of the form:

$$q_i(x_1^*, \ldots, x_{i-1}^*, x_i, x_{i+1}^*, \ldots, x_n^*) \leq q_i(x^*) \ .$$

This means applying a b&b method for each box – a b&b method in a reduced space, but still it sounds hopelessly inefficient and we have not implemented this solution.

The second possibility is to compare values of considered boxes during the b&b process. As it seems natural, it occurred to be extremely inefficient; traversing the list of all boxes in each iteration is very time consuming. Also, for parallel computations it requires not to change the queue of boxes while some threads are traversing it to compare criteria values. This solution was discarded, too.

The third possibility is to compare only values of the selected boxes in a "second phase". It is far more efficient than the previous variant, but still has a quadratic complexity with respect to the number of boxes. It has to be noted that – as this variant works perfectly for considered examples – it is weaker than the other ones; some boxes that would be discarded by previous algorithms would not be discarded now. Nevertheless, it is the reference variant in current implementation.

The fourth possibility is to have a specific data structure for each player, where the values they may obtain for specific inputs from other users will be stored. The check can be done either during the b&b process or in a "second phase", after finishing the b&b process. There is a very adequate data structure to store such information, called an interval tree (see [21]). This approach seems promising, but has not been implemented yet.

Rejection/reduction tests. Rejection/reduction tests are commonly used in interval branch-and-bound methods (see e.g. [2],[4], [5], [13]) In our case two tests are used; they were already mentioned above. They are:

- a variant of the monotonicity test,
- a Gauss-Seidel operator with the inverse-midpoint preconditioner, applied to solve equations system ([2]).

The overall algorithm. The following pseudocode expresses the main schema of the solver:

`seek_Nash_equilibria` ($\boldsymbol{x}^{(0)}$; $\mathsf{f}_1, \ldots, \mathsf{f}_n$)
// $\boldsymbol{x}^{(0)}$ is the initial box,
// $\mathsf{f}_i(\cdot)$ is the interval extension of the function $f_i \colon \mathbb{R}^N \to \mathbb{R}$, $i = 1, \ldots, n$
// L_{sol} is the list of boxes verified to contain a Nash point
$L = \emptyset$;
$L_{sol} = \emptyset$;
enqueue (L, $\boldsymbol{x}^{(0)}$) ;
`while` (L is nonempty)
    dequeue (L, $\boldsymbol{x}$) ;
    process the box $\boldsymbol{x}$, using the rejection/reduction tests ;
    `if` ($\boldsymbol{x}$ does not contain a solution) `then` discard $\boldsymbol{x}$;
    `else if` ($\boldsymbol{x}$ is verified to contain a solution) `then` enqueue (L_{sol}, $\boldsymbol{x}$) ;
    `else if` ($\mathrm{diam}(\boldsymbol{x}) \leq \varepsilon$) `then` enqueue ($L_{sol}$, $\boldsymbol{x}$) ;
    `else`
        bisect ($\boldsymbol{x}$), obtaining $\boldsymbol{x}^{(1)}$ and $\boldsymbol{x}^{(2)}$;
        $\boldsymbol{x} = \boldsymbol{x}^{(1)}$;
        enqueue (L, $\boldsymbol{x}^{(2)}$) ;
    `end if` ;
`end while`
// Second phase
`foreach` ($\boldsymbol{x}$ in L_{sol})
    `if` ($\boldsymbol{x}$ cannot contain a Nash equilibrium) `then` discard $\boldsymbol{x}$;
`end foreach`

4 Parallelization

As discussed in Section 3, the algorithm has two phases:

- the branch-and-bound phase, seeking points satisfying necessary conditions for Nash equilibria,
- discarding boxes that do not contain equilibria – comparisons between boxes, computed by the first phase.

As our previous experiments show ([8], [9]), using several threads can significantly increase the efficiency of the computations, so we parallelized the computations. Both phases are parallelized independently.

In the first phase boxes are stored in a queue and processed in parallel by a few threads. There is a clever implementation of a queue with two locks (see [11]) that allows parallel put and get operations on both ends of the queue. Specifically, one lock guards the access to the head of the list, another one – the tail and there is a dummy node at the head, so that the ultimate element could not be deleted during the insertion. Details can be found in [11].

In the second phase a few threads are traversing the list simultaneously to find if different boxes should be discarded or not.

POSIX threads [23] were used for the parallelization, but any other tool (specifically OpenMP) could also be used with minor changes only.

5 Numerical Experiments

We shall present results for three test problems – all of them first discussed in [19] and then considered in [3], [7].

The game has two players, each of them may control one decision variable.

$$\begin{aligned} &\min_{x_1} \left(q_1(x_1,x_2) = (x_1 - x_2 + 1)^2\right) \\ &\min_{x_2} \left(q_2(x_1,x_2) = (x_2 - x_1^2)^2 + (x_1 - 1)^2\right) , \\ &x_1 \in [-1, 2.5] \ , \ x_2 \in [-1, 3] \ . \end{aligned} \tag{3}$$

This game has three Nash equilibria: $(2, 3)$ on the boundary and two in the interior of the feasible set: $(-0.618034, 0.381966)$ and $(1.618033, 2.618033)$.

Second game is also a game of two players, but now each of them has 9 decision variables.

$$\begin{aligned} &\min_{x_1,\ldots,x_9} \left(q_1(x) = (x_1 - 1)^2 + (x_2 - 1)^2 + x_3^2 + (x_4 - 1)^2 + x_5^2 + (x_6 - 1)^2 + \right. \\ &\left. + (x_7 - 1)^2 + x_8^2 + x_9^2 + x_{11}^2 + (x_{12} - 0.5)^2 + x_{13}^2 + (x_{16} + 0.5)^2 + (x_{18} - 1)^2\right) , \\ &\min_{x_{10},\ldots,x_{18}} \left(q_2(x) = (x_{10} + 1)^2 + x_{11}^2 + (x_{12} - 1)^2 + x_{13}^2 + x_{14}^2 + (x_{15} + 1)^2 + \right. \\ &\left. + (x_{17} - 1)^2 + x_{16}^2 + (x_{18} - 1)^2 + (x_2 - 0.5)^2 + x_3^2 + (x_4 - 0.5)^2 + (x_8 - 0.5)^2\right) , \\ &x_i \in [-2, 2] \qquad i = 1, 2 \ . \end{aligned} \tag{4}$$

The game has one Nash equilibrium: $(1,1,0,1,0,1,1,0,0,-1,0,1,0,0,-1,0,1,1)$.

In the third game we have three players, with two decision variables each.

$$\min_{x_1,x_2} \left(q_1(x) = (x_1+1)^2(x_1-1)^2 + (x_2+1)^2(x_2-1)^2 + x_3x_4 + x_5x_6\right) \quad (5)$$

$$\min_{x_3,x_4} \left(q_2(x) = (x_4-0.5)^2(x_4+1)^2 + (x_3+1)^2 + x_1x_2 + x_5x_6\right) ,$$

$$\min_{x_5,x_6} \left(q_3(x) = (x_5+0.5)^2(x_5-1)^2 + (x_6-1)^2 + x_1x_2 + x_3x_4\right) ,$$

$$x_i \in [-2,2] \qquad i = 1,\ldots,6 .$$

This game has 16 Nash equilibria (they are listed in [7] and [19]).

Numerical experiments were performed on a computer with 16 cores, i. e. 8 Dual-Core AMD Opterons 8218 with 2.6GHz. The machine ran under control of a Fedora 10 Linux operating system. The solver was implemented in C++, using C-XSC 2.2.3 library for interval computations. The GCC 4.3.2 compiler was used.

Results are given in Tables 1–4.

For all experiments the accuracy $\varepsilon = 10^{-7}$ was set. Solutions computed for problem (3):

$$x = \left([-0.618034, -0.618033], [0.381966, 0.381967]\right) ,$$
$$x = \left([1.618033, 1.618034], [2.618033, 2.618034]\right) ,$$
$$x = \left([1.999999, 2.000001], [3.000000, 3.000000]\right) .$$

The solution computed for problem (4):

$$\begin{aligned} x = \big(&[1.000000, 1.000000], [1.000000, 1.000000], [-0.000000, 0.000000], \\ &[1.000000, 1.000000], [-0.000000, 0.000000], [1.000000, 1.000000], \\ &[1.000000, 1.000000], [-0.000000, 0.000000], [-0.000000, 0.000000], \\ &[-1.000000, -1.000000], [-0.000000, 0.000000], [1.000000, 1.000000], \\ &[-0.000000, 0.000000], [-0.000000, 0.000000], [-1.000000, -1.000000], \\ &[-0.000000, 0.000000], [1.000000, 1.000000], [1.000000, 1.000000]\big) . \end{aligned}$$

Table 1. Numerical results for test problem (3)

computational time (sec.)	0.014
criteria evals.	6
criteria grad. evals.	26
criteria Hesse matrix evals.	204
bisections	49
boxes bisected by GS step	0
boxes deleted by comparisons	0
boxes deleted by monot. test	42
boxes deleted by GS step	5
resulting boxes	3

Table 2. Numerical results for the test problem (4)

domain	original	$[-2, 2.4]^{18}$
computational time (sec.)	2483392	0.370
criteria evals.	524288	2
criteria grad. evals.	524288	2
criteria Hesse matrix evals.	18878160	182
bisections	4719540	45
boxes bisected by GS step	0	0
boxes deleted by comparisons	0	0
boxes deleted by monot. test	4457400	45
boxes deleted by GS step	0	0
resulting boxes	262144	1

First four solutions computed for problem (5):

$$x = \big([0.999999, 1.000001], [-1.000001, -0.999999], [-1.000000, -1.000000], [-1.000001, -0.999999], [0.999999, 1.000001], [1.000000, 1.000000]\big) ,$$

$$x = [0.999999, 1.000001], [-1.000001, -0.999999], [-1.000000, -1.000000], [-1.000001, -0.999999], [-0.500001, -0.499999], [1.000000, 1.000000]\big) ,$$

$$x = [-1.000000, -1.000000], [-1.000000, -1.000000], [-1.000000, -1.000000], [0.499999, 0.500001], [0.999999, 1.000001], [1.000000, 1.000000]\big) ,$$

$$x = [-1.000000, -1.000000], [-1.000000, -1.000000], [-1.000000, -1.000000], [0.499999, 0.500001], [-0.500001, -0.499999], [1.000000, 1.000000]\big) .$$

Table 3. Numerical results for test problem (5)

domain	original	$[-2, 2.4]^6$
computational time (sec.)	47.703	1.410
criteria evals.	13392	243
criteria grad. evals.	83088	2286
criteria Hesse matrix evals.	149925	6297
bisections	21827	683
boxes bisected by GS step	3160	366
boxes deleted by comparisons	3440	65
boxes deleted by monot. test	14748	870
boxes deleted by GS step	5776	99
resulting boxes	1024	16

Table 4. Numerical results of the parallelized algorithm for test problem (5)

threads num.	1	2	4	6	7	8
time (sec.)	48	24	12	9	10	9
speedup	1.00	2.00	4.00	5.33	4.80	5.33

6 Results

For test problem (3) the problem performed perfectly – the computations were extremely fast and it found precisely all three solutions; also the one on the boundary $(2, 3)$ that was missed by algorithms in [7] and [19]i, based on Nikaidô and Isoda transformation.

For test problems (4) and (5) the algorithm preformed worse, initially (left columns in Tables 2 and 3).

The reason was a bit unusual, but interesting – the solutions are located exactly in the corners of boxes resulting from the bisection process and hence they have to be enclosed by clusters of boxes. It is easy to see that for 18 dimensions the solution has to be enclosed by $2^{18} = 262144$ boxes, so all of the boxes enclose this one solution! Also for 6 dimensions we need $2^6 = 64$ boxes for one solution and $16 \cdot 64 = 1024$ for 16 of them; again that is the reason of the large number of boxes.

As we can see from Tables 2 and 3 increasing (!) the domain only slightly undoes this effect and improves the performance of the algorithm incredibly.

Comment. Actually, it is worth noting that this embarrassing difference (in case of problem (4) from several hours to a couple of milliseconds) shows two facts:

- how significant it is to choose test problems adequately,
- how sensitive interval methods are to this – unlikely, but possible – phenomenon of "unfortunate location" of solutions.

Possibly algorithms should pay more attention to the second problem, e. g. by doing some local search and and constructing boxes around such solutions.

7 Conclusions

As we have seen, interval methods can be quite efficient at finding all Nash equilibria of continuous games. As usually, they are also robust – the find also equilibrium points that are missed by most other algorithms. For the considered example (5), the parallelization worked perfectly for up to four threads; for a higher number of threads some improvement was still observed.

Still many improvements can be done to the algorithms, e. g. to include arbitrary constraints for the domain of decision variables.

In any case there is obviously one – probably impossible to overcome – limitation for the use of interval methods: they require that some central authority knows formulae for criteria of the players. In the situation when the central authority does not have such knowledge – or even she does not want the players to pass too much information, but only she observes their behavior – it is difficult to imagine application of interval methods; at least in their traditional form.

Nevertheless, we can imagine several situations where the criteria functions are given by known mathematical formulae; then there is no obstacle to apply interval methods.

Acknowledgments

The research has been supported by the Polish Ministry of Science and Higher Education under grant N N514 416934.

The computer on which experiments were performed is shared with the Institute of Computer Science of our University. Thanks to Jacek Błaszczyk for maintaining it.

References

1. Gabay, D., Moulin, H.: On the uniqueness and stability of Nash-equilibria in noncooperative games. In: Bensoussan, A., Kleindorfer, P., Tapiero, C. (eds.) Applied Stochastic Control in Econometrics and Management Science, pp. 271–293. North Holland, Amsterdam (1980)
2. Hansen, E., Walster, G.E.: Global Optimization Using Interval Analysis, Second edn. Revised and Expanded. Marcel Dekker, New York (2004)
3. Jauernig, K., Kołodziej, J., Stysło, M.: HGSNash evolutionary strategy as an effective method of detecting the Nash equilibria in n-person non-cooperative games. In: Proceedings of KAEiOG 2006, Murzasichle, pp. 171–178 (2006)
4. Jaulin, L., Kieffer, M., Didrit, O., Walter, E.: Applied Interval Analysis. Springer, London (2001)
5. Kearfott, R.B.: Rigorous Global Search: Continuous Problems. Kluwer, Dordrecht (1996)
6. Kearfott, R.B., Nakao, M.T., Neumaier, A., Rump, S.M., Shary, S.P., van Hentenryck, P.: Standardized notation in interval analysis, http://www.mat.univie.ac.at/~neum/software/int/notation.ps.gz
7. Kołodziej, J., Jauernig, K., Cieślar, A.: HGSNash strategy as the decision-making method for water resource systems with external disagreement of interests. In: Proceedings of PARELEC 2006, Wrocław, pp. 313–318 (2006)
8. Kubica, B.J.: Interval methods for solving underdetermined nonlinear equations systems. In: SCAN 2008 Conference, El Paso, Texas (2008)
9. Kubica, B.J., Woźniak, A.: A multi-threaded interval algorithm for the Pareto-front computation in a multi-core environment. In: PARA 2008 Conference, Trondheim, Norway (2008)
10. MacKenzie, A.B., Wicker, S.B.: Game theory in communications: Motivation, explanation, and application to power control. In: Proceedings of IEEE GLOBECOM 2001, pp. 821–826 (2001)
11. Michael, M.M., Scott, M.L.: Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In: Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, pp. 267–275 (1996)
12. Nash, J.F.: Equilibrium points in n-person games. Proceedings of National Association of Science 36, 48–49 (1950)
13. Neumaier, A.: Interval methods for systems of equations. Cambridge University Press, Cambridge (1990)
14. Nikaidô, H., Isoda, K.: Note on noncooperative convex games. Pacific Journal of Math. 5(Suppl. I), 807–815 (1955)
15. Pavlidis, N.G., Parsopoulos, K.E., Vrahatis, M.N.: Computing Nash equilibria through computational intelligence methods. Journal of Computational and Appl. Math. 175, 113–136 (2005)

16. Petrosyan, L.A., Zakharov, V.V.: Introduction to mathematical ecology. Izd. LGU, Leningrad (1986) (in Russian)
17. Rosen, J.B.: Existence and uniqueness of equilibrium points for concave n-person games. Econometrica 33, 520–534 (1965)
18. Serrano, R.: The theory of implementation of social choice rules. SIAM Review 46, 377–414 (2004)
19. Ślepowrońska, K.: A parallel algorithm for finding Nash equilibria (in Polish). Master's thesis under supervision of A. Woźniak, WUT (1996)
20. Nash equilibrium, DDWiki article, http://ddl.me.cmu.edu/ddwiki/index.php/Nash_equilibrium#Formal_Description
21. Interval tree, Wikipedia article, http://en.wikipedia.org/wiki/Interval_tree
22. C-XSC interval library, http://www.xsc.de
23. POSIX Threads Programming, https://computing.llnl.gov/tutorials/pthreads

From Gauging Accuracy of Quantity Estimates to Gauging Accuracy and Resolution of Measuring Physical Fields

Vladik Kreinovich[1] and Irina Perfilieva[2]

[1] University of Texas, El Paso, TX 79968, USA
vladik@utep.edu
[2] University of Ostrava, Inst. for Research and Applications of Fuzzy Modeling, 70100 Ostrava, Czech Republic
Irina.Perfilieva@osu.cz

Abstract. For a numerical physical quantity v, because of the measurement imprecision, the measurement result $\widetilde{v}$ is, in general, different from the actual value v of this quantity. Depending on what we know about the measurement uncertainty $\Delta v \stackrel{\text{def}}{=} \widetilde{v} - v$, we can use different techniques for dealing with this imprecision: probabilistic, interval, etc.

When we measure the values $v(x)$ of physical fields at different locations x (and/or different moments of time), then, in addition to the same measurement uncertainty, we also encounter another type of *localization* uncertainty: that the measured value may come not only from the desired location x, but also from the nearby locations.

In this paper, we discuss how to handle this additional uncertainty.

1 Formulation of the Problem

Need for data processing. In many real-life situations, we are interested in the value of a quantity which is difficult (or even impossible) to measure directly. For example, we may be interested in the distance to a star, or in the amount of water in an underground water layer. Since we cannot measure the corresponding quantity y directly, we measure it *indirectly*. Specifically,

- we find easier-to-measure quantities $x_1, \ldots, x_n$ which are related to the desired quantity y by a known dependence $y = f(x_1, \ldots, x_n)$;
- we measure the values of the auxiliary quantities $x_1, \ldots, x_n$; and
- we use the results $\widetilde{x}_1, \ldots, \widetilde{x}_n$ of measuring the auxiliary quantity to compute the estimate $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ for the desired quantity y.

Example. To find the distance y to a star, we can use the following *parallax* method:

- we measure the orientations x_1 and x_2 to this star at two different seasons,
- we measure the the distance x_3 between the spatial locations of the corresponding telescopes at these two seasons (i.e., in effect, the diameter of the earth orbit);

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 456–465, 2010.

– then, reasonably simply trigonometric computations enable us to describe the desired distance y as a function of the easier-to-measure quantities x_1, x_2, and x_3.

General case. In general, computations related to such indirect measurements form an important particular case of data processing.

Need to take uncertainty into account. Measurements are never absolutely accurate. As a result, the measurement results $\widetilde{x}_i$ are, in general, different from the actual (unknown) values x_i of the measured quantities: $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i \neq 0$. Because of this, the result $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_n)$ of data processing is, in general, different from the actual (unknown) value $y = f(x_1, \ldots, x_n)$: $\Delta y \stackrel{\text{def}}{=} \widetilde{y} - y \neq 0$.

Thus, in practical applications, we need to take this uncertainty into account.

Interval uncertainty. In practice, we often only know the upper bound Δ_i on the measurement errors $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$: $|\Delta x_i| \leq \Delta_i$. In this case, the only information that we have about the actual values x_i is that x_i belongs to the *interval* $\mathbf{x}_i \stackrel{\text{def}}{=} [\widetilde{x}_i - \Delta_i, \widetilde{x}_i + \Delta_i]$.

Under such interval uncertainty, we need to find the range of possible values of y: $\mathbf{y} = \{f(x_1, \ldots, x_n) : x_i \in \mathbf{x}_i\}$. The problem of computing this range is known as *interval computations*; see, e.g., [4].

Need to measure physical fields. In practice, the situation is often more complex: the values that we measure can be:

– values $v(t)$ of a certain dynamic quantity v at a certain moment of time t
– or, more generally, the values $v(x, t)$ of a certain physical field v at a certain location x and at a certain moment of time t.

For example, in geophysics, we are interested in the values of the density at different locations and at different depth.

Need to take uncertainty into account when measuring physical fields. When we measure physical fields, not only we get the measured value $\widetilde{v} \approx v$ with some inaccuracy, but also the location x is not exactly known. Moreover, the sensor picks up the "averaged" value of v at locations close to the approximately known location $\widetilde{x}$. In other words, in addition to inaccuracy $\widetilde{v} \neq v$, we also have a finite (spatial) *resolution* $\widetilde{x} \neq x$.

Estimating uncertainty related to measuring physical fields: challenging problems. In general, the measured value $\widetilde{v}_i$ differs from the averaged value v_i by the measurement imprecision $\Delta v_i = \widetilde{v}_i - v_i$. In the interval case, we know the upper bound Δ_i on this measurement error $|\Delta v_i| \leq \Delta_i$. Thus, the averaged quantity v_i can take any value from the interval $[\underline{v}_i, \overline{v}_i]$, where $\underline{v}_i \stackrel{\text{def}}{=} \widetilde{v}_i - \Delta_i$ and $\overline{v}_i \stackrel{\text{def}}{=} \widetilde{v}_i + \Delta_i$.

Based on these bounds on v_i, what can we learn about the original field $v(x)$? The answer to this questions depends on what we know about the averaging, i.e., on the dependence of v_i on $v(x)$. In principle, there are three possible situations:

– sometimes, we know exactly how the averaged values v_i are related to $v(x)$;
– sometimes, we only know the upper bound δ on the location error $\widetilde{x} - x$ (this is similar to the interval case);
– sometimes, we do not even know δ.

In the following sections, we describe how to process all these types of uncertainty.

2 Possibility of Linearization

Sometimes, the signal $v(x)$ that we are measuring is large, i.e., the values of the signal are much larger than the noise (and the measurement errors in general). In such situations, the measured values well represent the actual signal, and for many applications, the measurement errors can be safely ignored.

The need to take into account measurement errors becomes important only when the signal $v(x)$ is relatively weak. In this case, we can expand the dependence of v_i on $v(x)$ in Taylor series. To describe this expansion, let us first consider a simplified case in which there are only finitely many spatial points $x^{(1)}$, ..., $x^{(N)}$, so the field $v(x)$ is described by finitely many values $v^{(j)} \stackrel{\text{def}}{=} v\left(x^{(j)}\right)$, $j = 1, \ldots, N$. In this case, the dependence of the quantity v_i on these values $v^{(1)}$, ..., $v^{(N)}$ can be expanded into Taylor series

$$v_i = a_i + \sum_{j=1}^{N} a_{ij} \cdot v(x^{(j)}) + \sum_{j=1}^{N} \sum_{j=k}^{N} a_{ijk} \cdot v(x^{(j)}) \cdot v(x^{(k)}) + \ldots \tag{1}$$

The description in terms of finitely many spatial points is an approximate description of the actual field $v(x)$. The more points we consider and the denser they are located in the domain D on which this field is defined, the more accurate the corresponding approximation. In the limit, when the distance between the points tends to 0 (and the approximation accuracy tends to 0), the sums in (1) turn into integrals, so the formula (1) takes the form

$$v_i = a_i + \int_D w_i(x) \cdot v(x)\, dx + \int_{D \times D} w_i(x, x') \cdot v(x) \cdot v(x')\, dx\, dx' + \ldots \tag{2}$$

for appropriate functions $w_i(x)$, $w_i(x, x')$, etc.

Since the signal $v(x)$ is relatively weak, we can safely ignore quadratic and higher order terms in this dependence. Also, we know that in the absence of the field, when $v(x) = 0$, the differences v_i are 0s, so we have $a_i = 0$. As a result, we get a linear expression for v_i in terms of $v(x)$: $v_i = \int_D w_i(x) \cdot v(x)\, dx$.

3 Case of Full Information about the Resolution

Description. In this section, we consider the case when we know the exact expression for this dependence, i.e., when we know the weights $w_i(x)$.

The notion of fuzzy transform. Intuitively, each "averaged" value v_i can be viewed as the value of the field $v(x)$ at a "fuzzy" point characterized by uncertainty $w_i(x)$. Because of this interpretation, the transformation from the original function $v(x)$ to the set of values $v_1, \ldots, v_n$ is also known as a *fuzzy transform*; see, e.g., [6,7].

Comment. From the physical viewpoints, the weights $w_i(x)$ are not probabilities. However, since probability theory is the oldest – and most developed – formalism for describing uncertainty, it is not surprising that often, computational techniques from probability theory can be efficiently used to described other types of uncertainty as well. In particular, in the following text, we will see that for our problem, techniques from imprecise probability theory can be very useful.

What we want to predict. Based on the measurement results $\widetilde{v}_1, \ldots, \widetilde{v}_n$, we would like to reconstruct the field $v(x)$. From the pragmatic viewpoint, knowing the field means being able to predict the results of all other measurements of this field.

Each such measurement can be characterized by its own averaging function $w(x)$. Thus, predicting the result of the measurement means predicting the corresponding averaged value $y = \int_D w(x) \cdot v(x)\, dx$.

Of course, the space of functions is infinite-dimensional, which means that to uniquely reconstruct a function, we need to know infinitely many parameters. Thus, based on n numbers $\widetilde{v}_1, \ldots, \widetilde{v}_n$, we cannot uniquely reconstruct the function $v(x)$ – and thus, we cannot uniquely reconstruct the desired averaged value y. So, the problem is to find the *range* $[\underline{y}, \overline{y}]$ of this value y.

Prediction problem as a particular case of (infinite-dimensional) linear programming (LP). The lower endpoint $\underline{y}$ is the smallest possible value of y, the upper endpoint $\overline{y}$ is the largest possible value of y. Thus, the problems of finding the desired endpoints $\underline{y}$ and $\overline{y}$ can be formulated in the following optimization form: minimize (maximize) $y - \int_D w(x) \cdot v(x)\, dx$ under the constraints

$$\underline{v}_i \le \int_D w_i(x) \cdot v(x)\, dx \le \overline{v}_i, \quad 1 \le i \le n. \tag{3}$$

In both problems, we optimize the value of a linear functional under linear constraints. In the finite-dimensional case, when we have finitely many unknowns, such constraint optimization problems are known as linear programming (LP) problems. In our case, an unknown is a function $v(x)$, and the linear space of all possible functions is infinite-dimensional. Thus, from the mathematical viewpoint, our problems are infinite-dimensional analogues of linear programming (LP) problems; see, e.g., [1].

Without prior restrictions on the field $v(x)$, we cannot predict anything. In general, if we do not impose any conditions on the function $v(x)$, then both bounds

are infinite – unless $w(x)$ is a linear combination of $w_i(x)$. Indeed, it is known that every vector w which is orthogonal to all the vectors t, which are orthogonal to all the vectors $w_1, \ldots, w_n$, belongs to the linear space generated by the vectors $w_1, \ldots, w_n$ – i.e., is a linear combination of $w_1, \ldots, w_n$. Thus, if a vector w cannot be represented as a linear combination of the vectors $w_1, \ldots, w_n$, then there exists a vector t which is orthogonal to all w_i but not to w. With respect to the space of all the functions, this means that if $w(x)$ cannot be represented as a linear combination of the functions $w_i(x)$, then there exists a function $t(x)$ which is orthogonal to all $w_i(x)$ (in the sense that $\langle w_i, t\rangle \stackrel{\text{def}}{=} \int_D w_i(x) \cdot t(x)\, dx = 0$) but not to $w(x)$ ($\langle w, t\rangle \neq 0$).

For an arbitrary real number λ, instead of the actual field $v(x)$, we can now consider a new field $v_\lambda(x) \stackrel{\text{def}}{=} v(x) + \lambda \cdot t(x)$. For this new field $v_\lambda(x)$, the values of v_i are the same as for the original field $v(x)$ – and hence, satisfy the same inequalities. However, the new value y is equal to $y_\lambda = \langle w, v\rangle + \lambda \cdot \langle w, t\rangle$. Since $\langle w, t\rangle \neq 0$, for appropriate λ, we can get this value y_λ equal to any given real number. Thus, indeed, the smallest possible value of y is $\underline{y} = -\infty$, and the largest possible value of y is $\overline{y} = +\infty$.

Non-negative fields. In many practical problems, the field $v(x)$ can only have non-negative values $v(x) \geq 0$. For example, in geophysics, the density $v(x)$ cannot be negative. Under this additional restrictions, we already have non-trivial bounds $\underline{y}$ and $\overline{y}$.

Dual LP techniques. For solving these problems, we can use the experience of imprecise probabilities [5,9] where we have similar LP problems. In these problems, the unknown function $v(x)$ is the non-negative probability density function (pdf), and the observed values have the same form

$$v_i = \int_D w_i(x) \cdot x(x)\, dx. \tag{4}$$

For example, the second moment of the probability distribution with the pdf $v(x)$ has the form $\int_D x^2 \cdot v(x)\, dx$, so it has the desired form with the weight function $w_i(x) = x^2$.

In the imprecise probability theory, it is reasonable to ask what is, e.g., the interval of possible values of the third moment if we know values of (or bounds on) the first and the second moments. In mathematical terms, we thus arrive at the same infinite-dimensional LP problems as in our measurement cases.

According to the experience of imprecise probabilities, many efficient algorithms for solving the corresponding LP problems come from considering *dual* LP problems, i.e., by computing the range $[\underline{v}, \overline{v}]$, where

$$\underline{v} = \sup\left\{\sum y_i \cdot \underline{v}_i : \sum y_i \cdot w_i(x) \leq w(x)\right\}; \tag{5}$$

and $\overline{v} = \inf\left\{\sum y_i \cdot \overline{v}_i : w(x) \leq \sum y_i \cdot w_i(x)\right\}$.

Indeed, if $\sum y_i \cdot w_i(x) \leq w(x)$, then, by multiplying both sides of this inequality by $v(x) \geq 0$ and integrating over x, we conclude that $\sum y_i \cdot v_i \leq y$. Since we

know that $v_i \geq \underline{v}_i$, we thus get a lower bound for y: $y \geq \sum y_i \cdot \underline{v}_i$. Thus, y is larger than the largest of these bounds, i.e., $y \geq \underline{v}$. So, we can conclude that $\underline{y} \geq \underline{v}$. Similarly, we can conclude that $\overline{y} \leq \overline{v}$, i.e., that the dual LP interval $[\underline{v}, \overline{v}]$ is the enclosure for the desired range $[\underline{y}, \overline{y}]$.

Comments.

- For finite-dimensional LP problems, the dual interval is exactly equal to the original one.
- Our problems are easier than the imprecise probability ones, since the functions $w_i(x)$ are usually localized and thus, for each x, usually at most a few functions $w_i(x)$ differ from 0. This makes checking the sums easier.
- Checking the inequalities like $\sum y_i \cdot w_i(x) \leq w(x)$ is even easier in a practically important case of piece-wise linear functions $w_i(x)$ and $w(x)$. In this case, it is sufficient to check this inequality at endpoints of linearity intervals – then, due to linearity, it will be automatically true for all internal points as well.

4 Situations in Which We Only Know Upper Bounds

General idea. In other cases – similarly to the interval setting – we do not only know the upper bounds δ on the location error $\widetilde{x} - x$. A natural question is: when is a model $v(x)$ consistent with the given observation $(\widetilde{v}, \widetilde{x})$?

In this case, the measured value $\widetilde{v}$ is Δ-close to a convex combination of values $v(x)$ for x s.t. $\|x - \widetilde{x}\| \leq \Delta x$. Thus, $\underline{v}_\delta(\widetilde{x}) - \Delta \leq \widetilde{v} \leq \overline{v}_\delta(\widetilde{x}) + \Delta$, where:

$$\underline{v}_\delta(\widetilde{x}) \stackrel{\text{def}}{=} \inf\{v(x) : \|x - \widetilde{x}\| \leq \delta\}, \text{ and } \overline{v}_\delta(\widetilde{x}) \stackrel{\text{def}}{=} \sup\{v(x) : \|x - \widetilde{x}\| \leq \delta\}. \quad (6)$$

Case of interval models. In real life, we rarely have an *exact* model $v(x)$. Usually, we have *bounds* on $v(x)$, i.e., an interval-valued model $\mathbf{v}(x) = [v^-(x), v^+(x)]$. An observation $(\widetilde{v}, \widetilde{x})$ consistent with this "interval-valued" model if there exists a model $v(x) \in \mathbf{v}(x)$ which is consistent with this observation.

Since the values $\underline{v}_\delta$ and $\overline{v}_\delta$ monotonically depend on $v(x)$, this consistency leads to $\underline{v}_\delta^-(\widetilde{x}) - \Delta \leq \widetilde{v} \leq \overline{v}_\delta^+(\widetilde{x}) + \Delta$.

Relation to Hausdorff metric. In many practical problems, the field $v(x)$ continuously depends on x. For continuous functions, inf and sup on a bounded closed set $\{x : \|x - \widetilde{x}\| \leq \delta\}$ are attained at some value. Thus, the above criterion for consistency between a model and observations can be simplified.

Namely, in this case, the set $\widetilde{m}$ of all measurement results $(\widetilde{v}, \widetilde{x})$ is consistent with the model $v(x)$ if and only if

$$\forall (\widetilde{v}, \widetilde{x}) \in \widetilde{m}\, \exists (v(x), x) \in v^{-1}\, ((\widetilde{v}, \widetilde{x}) \text{ is } (\Delta, \delta)\text{-close to } (v(x), x)), \quad (7)$$

i.e., $|\widetilde{v} - v| \leq \Delta$ and $\|x - \widetilde{x}\| \leq \delta$. In this formula, $v^{-1} \stackrel{\text{def}}{=} \{(v(x), x) : x \in D\}$, i.e., v^{-1} is the inverse relation to the relation $v = \{(x, v(x)) : x \in D\}$ describing the function $v(x)$.

The notion of (Δ, δ)-closeness between points (v, x) and (v', x') can be formally described as $d((v,x),(v',x')) \le (\Delta, \delta)$, where

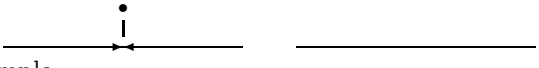
$$d((v,x),(v',x')) \stackrel{\text{def}}{=} (|v-v'|, \|x-x'\|);\ \ (\Delta,\delta) \le (\Delta',\delta') \Leftrightarrow ((\Delta \le \Delta')\,\&\,(\delta \le \delta')).$$

This definition is similar to the standard definition of the *Hausdorff metric* d_H: $d_H(A,B) \le \varepsilon$ means that

$$\forall a \in A\, \exists b \in B\, (d(a,b) \le \varepsilon) \text{ and } \forall b \in B\, \exists a \in A\, (d(a,b) \le \varepsilon). \tag{8}$$

(This similarity was noticed in [2].)

Specifically, the above definition is an *asymmetric two-dimensional* version of Hausdorff metric. Let us show, on a simple example, that our "distance" is indeed asymmetric.

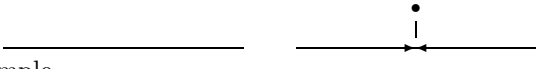
Case 1: 

In this example,

- the actual field has the form $v(0) = 1$ and $v(x) = 0$ for $x \ne 0$, and
- the measurements results are all zeros, i.e., $\widetilde{v} = 0$ for all $\widetilde{x}$.

In this case, all the measurements are consistent with the model:

- the values $\widetilde{v} = 0$ for $\widetilde{x} \ne 0$ are consistent with $v = 0$ for $x = \widetilde{x}$, and
- the value $\widetilde{v} = 0$ for $\widetilde{x} = 0$ is consistent with $v(x) = 0$ for $x = \delta$ s.t. $|\widetilde{x} - x| \le \delta$.

Case 2: 

In this example,

- the actual field is all zeros, i.e., $v(x) = 0$ for all x, and
- the measurement results are $\widetilde{v} = 1$ for $\widetilde{x} = 0$, and $\widetilde{v} = 0$ for all $\widetilde{x} \ne 0$.

Here, when $\Delta < 1$, the measurement $(1, 0)$ is *inconsistent* with the model, because for all x which are δ-close to $\widetilde{x} = 0$, we have $v(x) = 0$ hence we should have $|\widetilde{x} - v(x)| = |\widetilde{x}| \le \Delta$.

5 Case of Minimal Knowledge about Uncertainty

Idea. Yet another case is when we do not even know δ. It happened, e.g., when we solve the seismic inverse problem to find the velocity distribution. In this case, a natural heuristic idea is:

- to add a perturbation of size δ_0 (e.g., sinusoidal) to the reconstructed field $\widetilde{v}(x)$,
- to simulate the new measurement results,
- to apply the same algorithm to the simulated results, and
- to reconstruct the new field $\widetilde{v}_{\text{new}}(x)$.

If the perturbations are not visible in $\widetilde{v}_{\text{new}}(x) - \widetilde{v}(x)$, this means that details of size δ_0 cannot be reconstructed and so, the actual resolution is $\delta > \delta_0$. This approach was partially described in [3,8].

Linearization and its consequences. Which perturbations should we choose? To select the optimal perturbations, we will take into account the fact that since perturbations are usually small, we can safely linearize their effects. Thus, if we know the results $\Delta v_1(x), \ldots, \Delta v_k(x)$ of applying perturbations $e_1(x), \ldots, e_k(x)$, we can predict the result $\Delta v(x)$ of applying a linear combination

$$e(x) = c_1 \cdot e_1(x) + \ldots + c_k \cdot e_k(x), \tag{9}$$

as $\Delta v(x) = c_1 \cdot \Delta v_1(x) + \ldots + c_k \cdot \Delta v_k(x)$. In other words, once we know the results of applying k different perturbations $e_1(x), \ldots, e_k(x)$, we thus also know the results of applying an arbitrary perturbation from the linear space $L = \{c_1 \cdot e_1(x) + \ldots + c_k \cdot e_k(x)\}$. From this viewpoint, it does not matter what exactly perturbations $e_i(x)$ we select as long as they are within the same space L.

Thus, the question of optimally selecting a given number k of perturbations can be formulated as the question of optimally selecting a k-dimensional linear subspace L in the space of all functions.

Shift-invariance: a natural requirement. To select the space L, let us use the fact that in most physical problems, there is no preferred spatial location. Thus, in principle, we can choose different locations as origins ($x = 0$) of the coordinate system.

It is reasonable to require that the optimal family of perturbations do not change if we simply change the origin $x = 0$. For example, if we select a point with the original coordinates x_0 as the origin of a new coordinate system, then the new coordinates will have the form $x_{\text{new}} = x - x_0$. In the original coordinates, the optimal family of perturbations has the form $\{c_1 \cdot e_1(x) + \ldots + c_k \cdot e_k(x)\}$. In the new coordinates x_{new}, we should expect the exact same family of perturbations $\{c_1 \cdot e_1(x_{\text{new}}) + \ldots + c_k \cdot e_k(x_{\text{new}})\}$. In terms of the original coordinates, this new family has the form $\{c_1 \cdot e_1(x - x_0) + \ldots + c_k \cdot e_k(x - x_0)\}$.

This "shifted" family must coincide with the original one. In particular, every basis function $e_i(x - x_0)$ from the shifted basis must belong to the original family, i.e., must have the form $e_i(x + x_0) = \sum_{j=1}^{k} c_{ij}(x_0) \cdot e_j(x)$ for some coefficients c_{ij} which are, in general, depending on the shift x_0.

Smoothness: an additional requirement. In many physical problems, it is reasonable to consider smooth perturbations, i.e., perturbations for which the functions $e_i(x)$ are differentiable. In this case, by considering different values x, we get a system of linear equations for determining $c_{ij}(x_0)$ in terms of the smooth functions $e_i(x + x_0)$ and $e_j(x)$. The solution of a system of linear equations is – due to Cramer's rule – a smooth function of the coefficients and of the right-hand sides. Thus, the solutions $c_{ij}(x_0)$ are also smooth.

From the requirements to the description of the desired family L. Let us fix one of the spatial coordinates, e.g., the coordinate x_1. For shifts w.r.t. this coordinate, we have $e_i(x_1 + x_0, x_2, \ldots) = \sum_{j=1}^{k} c_{ij}(x_0) \cdot e_j(x_1, x_2, \ldots)$

Since the functions $e_i(x_1+x_0,\ldots)$ and $c_{ij}(x_0)$ are smooth, we can differentiate both sides of the above equation with respect to x_0 and take $x_0 = 0$. For each components of x_0, we get a system of linear differential equations $e_i' = \sum c_{ij}'(0)\cdot e_j$ with constant coefficients. A general solution to such a system is well known: it is a linear combination of expressions $x_1^{k_{1j}} \cdot \exp(a_{1j}\cdot x_1)$ with complex values a_{1j} (eigenvalues of the matrix $c_{ij}'(0)$) and integers $k_{1j} \geq 0$ (multiplicities of these eigenvalues).

Some of these solutions tend to infinity exponentially fast. Such solutions are not useful as perturbations, since perturbations must be uniformly small. So, it is reasonable to restrict ourselves to *bounded* perturbations.

This boundedness eliminates the terms with $\mathrm{Re}(a_{1j}) \neq 0$. Thus, the only remaining terms correspond to imaginary values a_{1j} – i.e., to sinusoids. For these terms, boundedness also eliminates terms with $k_{1j} > 0$, so we only get pure sinusoids: $e_i(x_1, x_2, \ldots) = \sum\limits_j C_j(x_2,\ldots)\cdot\sin(\omega_{1j}\cdot x_1)$. The functions $C_j(x_2,\ldots)$ can be computed as linear combinations of the values $e_i(x_1, x_2, \ldots)$ corresponding to different values x_1. On the other hand, the dependence of e_i on x_2 is also a linear combination of sinusoids. Thus, the functions $C_j(x_2,\ldots)$ are linear combinations of sinusoids in x_2. Substituting these linear combinations instead of $C_j(x_2,\ldots)$ into the above formula, and taking into account that $\sin(a)\cdot\sin(b)$ is a linear combination of $\cos(a+b)$ and $\cos(a-b)$, we conclude that the dependence of e_i on x_1 and x_2 takes the form

$$e_i(x_1, x_2, x_3, \ldots) = \sum_k C_k(x_3,\ldots)\cdot\sin(\omega_{1k}\cdot x_1 + \omega_{2k}\cdot x_2). \tag{10}$$

Similarly, we can add x_3, etc., and conclude that each function $e_i(x)$ is a linear combination of the sinusoids $\sin(\sum \omega_j \cdot x_j + \varphi)$.

Resulting recommendation. We conclude that the optimal perturbations are linear combinations of sinusoids. We thus arrive at the following recommendation: to find the spatial resolution δ with which we can reconstruct the field $v(x)$, add a sinusoid with spatial wavelength δ_0 to the reconstructed field $\widetilde{v}(x)$, simulate the new measurement result, reconstruct the new field $\widetilde{v}_{\mathrm{new}}(x)$, and see if the perturbations are visible in $\widetilde{v}_{\mathrm{new}}(x) - \widetilde{v}(x)$.

6 Conclusions

When we measure the values $v(x)$ of physical fields at different locations x (and/or different moments of time), then, in addition to the measurement uncertainty, we also encounter another type of *localization* uncertainty: that the measured value may come not only from the desired location x, but also from the nearby locations. In this paper, we discuss how to handle this additional uncertainty. Specifically:

– in situations in which we know the exact dependence of the measured value on the field $v(x)$, we can use infinite-dimensional versions of linear programming techniques;

– in situations in which we only know upper bounds on localization errors, we can use 2-dimensional versions of Hausdorff metric; and
– in situations in which we have no information about localization uncertainty, we can use sinusoidal perturbations to acquire this information.

Acknowledgments. This work was supported in part by the National Science Foundation grant HRD-0734825, by Grant 1 T36 GM078000-01 from the National Institutes of Health, and by Grant MSM 6198898701 from MŠMT of Czech Republic. The authors are thankful to the anonymous referees and to the participants of PPAM'09 for valuable suggestions.

References

1. Anderson, E.J., Nash, P.: Linear Programming in Infinite-Dimensional Spaces. Wiley, New York (1987)
2. Anguelov, R., Markov, S., Sendov, B.: The set of Hausdorff continuous functions – the largest linear space of interval functions. Reliable Computing 12, 337–363 (2006)
3. Averill, M.G.: A Lithospheric Investigation of the Southern Rio Grande Rift, PhD Dissertation, University of Texas at El Paso, Department of Geological Sciences (2007)
4. Jaulin, L., et al.: Applied Interval Analysis. Springer, London (2001)
5. Kuznetsov, V.: Interval Statistical Methods. Radio i Svyaz Publ., Moscow (1991) (in Russian)
6. Perfilieva, I.: Fuzzy transforms: theory and applications. Fuzzy Sets and Systems 157, 993–1023 (2006)
7. Perfilieva, I., Novák, V., Dvorák, A.: Fuzzy transform in the analysis of data. International Journal of Approximate Reasoning 48(1), 36–46 (2008)
8. Pinheiro da Silva, P., et al.: Propagation and Provenance of Probabilistic and Interval Uncertainty in Cyberinfrastructure-Related Data Processing and Data Fusion. In: Muhanna, R.L., Mullen, R.L. (eds.) Proceedings of the International Workshop on Reliable Engineering Computing REC 2008, Savannah, Georgia, February 20-22, pp. 199–234 (2008)
9. Walley, P.: Statistical Reasoning with Imprecise Probabilities. Wiley, Chichester (1991)

A New Method for Normalization of Interval Weights

Pavel Sevastjanov, Pavel Bartosiewicz, and Kamil Tkacz

Institute of Comp. & Information Sci., Czestochowa University of Technology,
Dabrowskiego 73, 42-200 Czestochowa, Poland
sevast@icis.pcz.pl

Abstract. A new method for normalization of interval weights based on the so-called "interval extended zero" method is proposed. The three desirable intuitively obvious properties of normalization procedure are defined. The main of them is based on the assumption that the sum of normalized interval weights should be an interval centered around 1 with a minimal width. The advantages of a new method are illustrated with use of six numerical examples. It is shown that a new method performs better than known methods for normalization of interval weights as it provides the results with the properties which are close to the desirable ones.

Keywords: interval weights, normalization.

1 Introduction

The problem of interval and fuzzy weights normalization is of perennial interest, as it is related to the problems of Multiple Criteria Decision Making ($MCDM$) and the Depmpster-Shafer theory of evidence in the interval and fuzzy settings. For example, when the interval or fuzzy AHP is used in $MCDM$, the interval and fuzzy weights obtained from an interval or fuzzy pairwise comparison matrix should be normalized [1,2,16,19]. Moreover, normalized interval or fuzzy weights also facilitate the computation of interval and fuzzy weighted average [5,8,7,10,11]. In the Dempster-Shafer theory, imprecise evidence such as interval or fuzzy belief structures [4,17,20] also needs to be normalized. Therefore, the development of appropriate methods for interval and fuzzy weighs normalization is an important problem.

Let us consider the known methods for interval weights normalization. These methods are usually based on interval or fuzzy arithmetic. In the interval setting, the use of conventional interval arithmetic leads to the following reasoning [19].

Let $[w_i] = [w_i^L, w_i^U]$ and $[\hat{w}_i] = [\hat{w}_i^L, \hat{w}_i^U]$, $i = 1, \ldots, n$ be non-normalized and normalized interval weights, respectively. Then

$$[\hat{w}_i^L, \hat{w}_i^U] = \frac{[w_i]}{\sum_{j=1}^{n} [w_j]} = \left[\frac{w_i^L}{\sum_{j=1}^{n} w_j^U}, \frac{w_i^U}{\sum_{j=1}^{n} w_j^L} \right], i = 1, \ldots, n. \quad (1)$$

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 466–474, 2010.

It is known that expression (1) produces too wide normalized intervals, i.e., intervals $[\hat{w}_i]$ usually are substantially wider than initial non-normalized intervals $[w_i^L, w_i^U]$. It is seen that expression (1) is a formal interval extension of the normalization procedure used in the case of real valued weights. It is important that in the case of real valued weights, the normalization provides the weights, the sum of which is equal to 1, whereas in the interval setting, the equality $\sum_{i=1}^{n} [\hat{w}_i] = 1$ seems to be senseless as it holds only in the case of degenerated intervals ($\hat{w}_i^L = \hat{w}_i^U$, $i = 1, \ldots, n$) or when at least one of intervals $[\hat{w}_i]$ is inverted ($\hat{w}_i^L > \hat{w}_i^U$).

So the properties of interval normalization should be defined. Such properties which can be treated also as the definition of normalization are proposed in [18] as follows.

Let $N = \left\{ X = (x_1, \ldots, x_n) | \hat{w}_i^L \le x_i \le \hat{w}_i^U, i = 1, \ldots, n, \sum_{i=1}^{n} x_i = 1 \right\}$ be a set of normalized weight vectors. Then the interval weight vector $[\hat{w}] = ([\hat{w}_1^L, \hat{w}_1^U], \ldots, [\hat{w}_n^L, \hat{w}_n^U])$ is said to be normalized if and only if it satisfies the following two conditions:

(1). There exists at least one normalized weight vector $X = (x_1, \ldots, x_n)$ in the set N (N is none empty);
(2) All $\hat{w}_i^L$ and $\hat{w}_i^U$, $i = 1, \ldots, n$ are attainable in N.

It is shown in [18] that condition (1) is fulfilled if $\sum_{i=1}^{n} \hat{w}_i^L \le 1$ and $\sum_{i=1}^{n} \hat{w}_i^U \ge 1$.

Condition (2) means that each bound of $[\hat{w}_i]$, $i = 1, \ldots, n$ is attained for at least one vector in N, i.e., there exists such $X \in N$ that $x_j = \hat{w}_i^L$, $x_i \le \hat{w}_i^U, i = 1, ..., n, i \ne j$ and $x_k = \hat{w}_i^L$, $x_i \ge \hat{w}_i^L, i = 1, ..., n, i \ne k$ with $\sum_{i=1}^{n} x_i = 1$. The violation of condition (1) leads to the violation of condition (2), but the reverse is not always true. The above two conditions are fulfilled if $\sum_{i=1}^{n} \hat{w}_i^L + \max_j(\hat{w}_j^U - \hat{w}_j^L) \le 1$, $\sum_{i=1}^{n} \hat{w}_i^U - \max_j(\hat{w}_j^U - \hat{w}_j^L) \ge 1$. It is show in [18] that if for an initial interval weight vector the condition (1) is violated, then it can be normalized with use of the following model:

$$\min / \max w_i = z_i / \sum_{j=1}^{n} z_j,\ s.t. w_j^L \le z_j \le w_j^U,\ j = 1, ..., n.$$

This model results in the expression [18]:

$$[\hat{w}_i] = [\hat{w}_i^L, \hat{w}_i^U] = \left[\frac{w_i^L}{w_i^L + \sum\limits_{j \ne i} w_j^U}, \frac{w_i^U}{w_i^U + \sum\limits_{j \ne i} w_j^L} \right]. \quad (2)$$

The following model is proposed in [18] when only the condition (2) is violated:

$$\min / \max \hat{w}_i,\ s.t. w_j^L \le \hat{w}_j \le w_j^U, j = 1, ..., n,\ \sum_{j=1}^{n} \hat{w}_j = 1.$$

This model is a particular case of the possibility distribution model proposed by Dubois and Prade [6] and leads to the expression:

$$[\hat{w}_i] = [\hat{w}_i^L, \hat{w}_i^U] = \left[\max \left(w_i^L, 1 - \sum_{j \neq i} w_j^U \right), \min \left(w_i^U, 1 - \sum_{j \neq i} w_j^L \right) \right]. \quad (3)$$

Conditions (1) and (2) are rather of mathematical nature and do not form an exhaustive set of desirable properties of the interval weights normalization.

Our intension is to introduce such properties of interval weights normalization which are close to those of the normalization of real valued weights.

The first such property can be formulated as follows. As the sum of normalized real valued weights is always equal to 1, it seems natural to require $\sum_{i=1}^{n} [\hat{w}_i]$ ="near 1", where "near 1" may be presented by an interval.

The second desirable property of the interval weights normalization may be the remaining of ratios between the means of normalized intervals as close as possible to those of initial intervals as the normalization of real valued weights does not change these ratios at all. It is intuitively obvious also that after a proper normalizing procedure the ratios between interval lengths should be close enough to those before normalization. This is the third desirable property.

In the current report, we propose a new method for normalizing interval weights which provides the results with above three properties.

The rest of the paper is set out as follows. In Section 2, we recall the basics of the so-called "interval extended zero" method for the solution of linear interval equations [14,15] and show that it can be used for the normalization of interval weights. Section 3 is devoted to the comparison of the results obtained with use of approaches developed in [18] and a new method proposed in the current paper. Section 4 concludes with some remarks.

2 The Use of "Interval Extended Zero" Method for the Interval Weights Normalization

As the equation $\sum_{i=1}^{n} [\hat{w}_i] = 1$ is not verified for positive regular intervals $[\hat{w}_i]$, $i = 1, \ldots, n$, we propose to find such interval normalization factor $[x]$ that

$$\sum_{i=1}^{n} [\hat{w}_i][x] = \text{"}near\ 1\text{"}, \quad (4)$$

where "near 1" may be presented by an interval too. The more precise definition of "near 1" may be obtained using the following reasoning [14,15]: "Without loss

of generality, we can define the degenerated (usual) zero as the result of operation a-a, where a is any real valued number or variable. Hence, in a similar way we can define an "interval zero" as the result of operation $[a]$ - $[a]$, where $[a]$ is an interval. It is easy to see that for any interval $[a]$ we get $[\underline{a}, \overline{a}]$-$[\underline{a}, \overline{a}]$=$[\underline{a}-\overline{a}, \overline{a}-\underline{a}]$. Therefore, in any case, the result of interval subtraction $[a]$-$[a]$ is an interval centered around 0." Similarly we can define the "near 1" as an interval centered around 1.

The problem (4) can be solved with use of "interval extended zero" method [14,15].

Let us denote $[a] = \sum_{i=1}^{n} [\hat{w}_i]$. Then the problem $\sum_{i=1}^{n} [\hat{w}_i] = 1$ can be presented in the more general form:

$$[a][x] - [b] = 0, \tag{5}$$

where $[a]$, $[b]$ are positive intervals or real values. Of course, when $[a]$ or $[b]$ or both are intervals, the equation (5) has no solution as in the left hand side we have interval, whereas the right hand side is the degenerated zero (non-interval value). It is shown in [14,15] that if we treat the equation (5) as the result of interval extension of usual real valued equation $ax - b = 0$ then the right hand side of this equation (zero) should be extended too using the above definition of "interval zero". It must be emphasized that this definition says nothing about the width of "interval zero". Really, when extending equation with previously unknown values of variables in the left hand side, only what we can say about the right hand side is that it should be interval symmetrical with respect to 0 with not defined width. Hence, as the result of interval extension of $ax - b = 0$ in general case we get

$$[\underline{a}, \overline{a}][\underline{x}, \overline{x}] - [\underline{b}, \overline{b}] = [-y, y]. \tag{6}$$

Of course, the value of y in Eq.(6) is not yet defined and this seems to be quite natural since the values of $\underline{x}$, $\overline{x}$ are also not defined. In the case of positive intervals $[a]$ and $[b]$, i.e., $\underline{a}, \overline{a}, \underline{b}, \overline{b} > 0$ from Eq. (6) we get

$$\begin{cases} \underline{ax} - \overline{b} = -y, \\ \overline{ax} - \underline{b} = y. \end{cases} \tag{7}$$

Finally, from Eq. (7) we obtain only one linear equation with two unknown variables $\underline{x}$ and $\overline{x}$:

$$\underline{ax} + \overline{ax} - \underline{b} - \overline{b} = 0. \tag{8}$$

If there are some constraints on the values of unknown variables $\underline{x}$ and $\overline{x}$, then Eq. (8) with these constraints may be considered as the so called Constraint Satisfaction Problem (CSP) [3] and an interval solution may be obtained. The first constraint on the variables $\underline{x}$ and $\overline{x}$ is a solution of Eq. (8) assuming $\underline{x} = \overline{x}$. In this degenerated case, we get the solution of Eq. (8) as $x_m = \frac{\underline{b}+\overline{b}}{\underline{a}+\overline{a}}$. It is easy to see that x_m is the upper bound for $\underline{x}$ and the lower bound for $\overline{x}$ (if $\underline{x} > x_m$ or $\overline{x} < x_m$ we get an inverted solution of Eq. (8)). The natural low bound for $\underline{x}$ and upper bond for $\overline{x}$ may be defined using basic definitions of interval arithmetic [9,13] as $\underline{x}$=$\frac{\underline{b}}{\overline{a}}$, $\overline{x} = \frac{\overline{b}}{\underline{a}}$.

Thus, we have $[\underline{x}] = [\frac{\underline{b}}{\overline{a}}, x_m]$ and $[\overline{x}] = [x_m, \frac{\overline{b}}{\underline{a}}]$. These intervals can be narrowed taking into account Eq. (8), which in the spirit of CSP is treated as a constraint.

It is clear that the right bound of $\underline{x}$ and the left bound of $\overline{x}$, i.e., x_m, can not be changed as they present the degenerated (crisp) solution of (8). So let us focus of the left bound of $\underline{x}$ and right bound of $\overline{x}$.

From (8) we have

$$\underline{x} = \frac{\underline{b} + \overline{b} - \overline{a}\overline{x}}{\underline{a}}, \overline{x} \in [x_m, \frac{\overline{b}}{\underline{a}}], \overline{x} = \frac{\underline{b} + \overline{b} - \underline{a}\underline{x}}{\overline{a}}, \underline{x} \in [\frac{\underline{b}}{\overline{a}}, x_m]. \tag{9}$$

It is shown in [15] that for the positive $[a]$ and $[b]$, Eq. (9) have the following interval solution:

$$[\underline{x}] = \left[\underline{x}_{\max}, \frac{\underline{b} + \overline{b}}{\underline{a} + \overline{a}}\right], [\overline{x}] = \left[\frac{\underline{b} + \overline{b}}{\underline{a} + \overline{a}}, \overline{x}_{\min}\right], \tag{10}$$

where $\underline{x}_{\max} = \frac{\underline{b}}{\overline{a}}$, $\overline{x}_{\min} = \frac{\underline{b}+\overline{b}}{\underline{a}} - \frac{\underline{a}\underline{b}}{\overline{a}^2}$. Expressions (10) define all possible solutions of Eq. (6). The values of $\overline{x}_{min}, \underline{x}_{max}$ constitute an interval which produce the widest interval zero after substitution of them in Eq. (6). In other words, the maximum interval solution's width $w_{\max} = \overline{x}_{\min} - \underline{x}_{\max}$ corresponds to the maximum value of y: $y_{\max} = \frac{\overline{a}\overline{b}}{\underline{a}} - \underline{b}$. Substitution of the degenerated solution $\underline{x} = \overline{x} = x_m$ in Eq. (6) produces the minimum value of y: $y_{min} = \frac{\overline{a}\cdot\overline{b} - \underline{a}\cdot\underline{b}}{\underline{a}+\overline{a}}$. Obviously, for any permissible solution $\underline{x}' > \underline{x}_{max}$ there exists corresponding $\overline{x}' < \overline{x}_{min}$, for each $\underline{x}'' > \underline{x}'$ the inequalities $\overline{x}'' < \overline{x}'$ and $y'' < y'$ take place. Thus, the formal interval solution (10) factually represents the continuous set of nested interval solutions of Eq.(6) which can be naturally interpreted as a fuzzy number [14,15]. It is seen that values of y characterize the closeness of the right hand side of Eq. (6) to the degenerated zero and the minimum value y_{min} is defined exclusively by the interval parameters $[a]$ and $[b]$. Hence, the values of y may be considered, in a certain sense, as a measure of interval solution's uncertainty caused by the initial uncertainty of Eq. (6). Therefore, the following expression was introduced in [14,15]:

$$\alpha = 1 - \frac{y - y_{min}}{y_{max} - y_{min}}, \tag{11}$$

which may be treated as a certainty degree of interval solution of Eq. (6). We can see that α rises from 0 to 1 with decreasing of he interval's width from the maximum value to 0, i.e., with increasing of the solution's certainty. Consequently, the values of α may be treated as labels of α-cuts representing some fuzzy solution of Eq. (6). Finally, the solution is obtained in [14,15] in the form of triangular fuzzy number

$$\tilde{x} = \left\{\underline{x}_{\max}, \frac{\underline{b} + \overline{b}}{\underline{a} + \overline{a}}, \overline{x}_{\min}\right\}. \tag{12}$$

Obviously, we can assume the support of obtained fuzzy number to be a solution of analyzed problem. Such a solution may be treated as the "pessimistic" one since it corresponds to the lowest α-cuts of resulting fuzzy value. The word "pessimistic"is used here to emphasize that this solution is charged with the largest imprecision as it is obtained in the most uncertain conditions possible on the set of considered α -cuts. On the other hand, it seems natural to utilize

all additional information available in the fuzzy solution. The resulting fuzzy solution can be reduced to the interval solution using well known defuzzification procedures. In the considered case, the defuzzified left and right boundaries of the solution can be represented as

$$\underline{x}_{def} = \frac{\int_0^1 \underline{x}(\alpha)d\alpha}{\int_0^1 d\alpha}, \overline{x}_{def} = \frac{\int_0^1 \overline{x}(\alpha)d\alpha}{\int_0^1 d\alpha} \tag{13}$$

In the case of $[a], [b] > 0$, from (6) and (11) the expressions for $\underline{x}(\alpha)$ and $\overline{x}(\alpha)$ can be obtained. Substituting them into (13) we get

$$\underline{x}_{def} = \frac{\overline{b}}{\underline{a}} - \frac{y_{max} + y_{min}}{2\underline{a}}, \overline{x}_{def} = \frac{\underline{b}}{\overline{a}} + \frac{y_{max} + y_{min}}{2\overline{a}}. \tag{14}$$

It is shown in [14,15] that the proposed method provides the considerable reducing of resulting interval's length in comparison with that obtained using conventional interval arithmetic rules.

Let us turn to the normalization problem. To obtain the interval normalization factor $[x]$ in Eq. (4), we rewrite Eq. (6) as follows:

$$\sum_{i=1}^{n} [w_i] \cdot [x] - [1, 1] = [-y, y].$$

The solution is obvious and can be obtained substituting $\sum\limits_{i=1}^{n} [w_i]$ instead of $[a]$ and 1 instead of $\underline{b}$, $\overline{b}$, in (10). As the result we get

$$[x] = \left[\frac{1}{\sum\limits_{i=1}^{n} w_i^U}, \frac{2}{\sum\limits_{i=1}^{n} w_i^U} - \frac{\sum\limits_{i=1}^{n} w_i^L}{\left(\sum\limits_{i=1}^{n} w_i^U\right)^2} \right]. \tag{15}$$

Thus, the normalization procedure can be presented as

$$[\hat{w}_i] = [w_i] \cdot [x], i = 1, ..., n. \tag{16}$$

It is easy to see that for normalized weights $[\hat{w}_i]$, $i = 1, ..., n$ always $\sum\limits_{i=1}^{n} [\hat{w}_i] \subset [0, 2]$ and the sum $\sum\limits_{i=1}^{n} [\hat{w}_i]$ is an interval centered around 1. In our case, the expressions (14) take the form:

$$[x]_{def} = \left[\frac{1}{\sum\limits_{i=1}^{n} w_i^L} - \frac{y_{\max} + y_{\min}}{2\sum\limits_{i=1}^{n} w_i^L}, \frac{1}{\sum\limits_{i=1}^{n} w_i^U} + \frac{y_{\max} + y_{\min}}{2\sum\limits_{i=1}^{n} w_i^U} \right], \tag{17}$$

where $y_{\max} = 1 - \frac{\sum\limits_{i=1}^{n} w_i^L}{\sum\limits_{i=1}^{n} w_i^U}$, $y_{\min} = \frac{\sum\limits_{i=1}^{n} w_i^U - \sum\limits_{i=1}^{n} w_i^L}{\sum\limits_{i=1}^{n} w_i^L + \sum\limits_{i=1}^{n} w_i^U}$.

The normalization is presented as follows:

$$[\hat{w}_i] = [w_i] \cdot [x]_{def}, i = 1, ..., n. \tag{18}$$

3 The Comparison of a New Method with the Known Approaches for Interval Weights Normalization

As a base of comparison, we analyze here the six interval weights vectors numbered as follows:

1. $[w] = [0.06, 0.99], [0.1, 0.11], [0.5, 0.55]$.
2. $[w] = [0.3, 0.4], [0.3, 0.7], [0.4, 0.5], [0.5, 0.6]$.
3. $[w] = [0.3, 0.6], [0.2, 0.4], [0.1, 0.2]$.
4. $[w] = [0.01, 0.99], [0.01, 0.7], [0.01, 0.5]$.
5. $[w] = [2, 3], [3, 4], [1, 5]$.
6. $[w] = [3, 5], [2, 6], [7, 10], [1, 6], [3, 7], [6, 7]$.

The numbers of the above examples correspond to the numbers in Table 1 and Table 2. In Table 1, the sums of normalized interval weights $\sum_{i=1}^{n} [\hat{w}_i]$ obtained with use of the known methods (expressions (1),(2),(3)) and a new method (expressions 16,18) are presented.

In the examples 2,5 and 6, the condition 1 (see Section 1) is not fulfilled. Therefore, the expression (3) can not be used and the corresponding entries in Table 1 and Table 2 are empty.

It is seen that only a new method (expressions (16) and (18)) provides results such that $\sum_{i=1}^{n} [\hat{w}_i] =$ "$near\,1$", where "$near\,1$" is an interval centered around 1. Thus, only a new method is characterized by the first desirable property defined in Section 1. Moreover, a new method, especially expression (18), provides substantially more narrow sums $\sum_{i=1}^{n} [\hat{w}_i]$ than known approaches.

To analyze the other two desirable properties defined in Section 1, the corresponding measures of closeness of compared ratios were introduced.

Let $m_i = \frac{w_i^L + w_i^U}{2}$, $l_i = w_i^U - w_i^L$, $i = 1, ..., n$ be the means and lengths of non-normalized weights and $\hat{m}_i$, $\hat{l}_i$ be the means and lengths of normalized weights.

Table 1. The sums of normalized weights

N	$Non-normalized$	Exp. 1	Exp. 2	Exp. 3	Exp. 16	Exp. 18
1	[0.66, 1.65]	[0.4, 2.5]	[0.46, 1.56]	[0.94, 1.06]	[0.4, 1.6]	[0.49, 1.51]
2	[1.5, 2.2]	[0.68, 1.47]	[0.74, 1.31]		[0.68, 1.32]	[0.75, 1.25]
3	[0.6, 1.2]	[0.5, 2]	[0.62, 1.45]	[0.7, 1.2]	[0.5, 1.5]	[0.58, 1.42]
4	[0.03, 2.19]	[0.01, 73]	[0.02, 2.91]	[0.03, 2.18]	[0.01, 1.99]	[0.02, 1.98]
5	[6, 12]	[0.5, 2]	[0.58, 1.5]		[0.5, 1.5]	[0.58, 1.42]
6	[32, 61]	[0.52, 1.91]	[0.55, 1.73]		[0.52, 1.48]	[0.61, 1.39]

Table 2. The aggregated measures σ_m and σ_l

N	Exp. 1		Exp. 2		Exp. 3		Exp. 16		Exp. 18	
	σ_m	σ_l	σ_m	σ_l	σ_m	σ_l	σ_m	σ_l	σ_m	σ_l
1	1.036	46.63	1.095	51.67	0.869	51.22	0.563	42.95	0.27	36.154
2	0.035	1.139	0.028	1.417			0.026	1.008	0.013	0.705
3	0	0	0.219	0.821	0.215	0.646	0	0	0	0
4	0.012	0.013	0.629	0.657	0.014	0.014	0.006	0.007	0.003	0.007
5	0.114	0.239	0.135	0.406			0.074	0.174	0.038	0.104
6	0.121	0.992	0.079	1.037			0.079	0.866	0.04	0.65

Then the aggregated measure of closeness of ratios $\frac{m_i}{m_j}$ to ratios $\frac{\hat{m}_i}{\hat{m}_j}$ may be presented as $\sigma_m = \frac{1}{n(n-1)} \sqrt{\sum_{i=1}^{n} \sum_{j \neq i=1}^{n} \left(\frac{m_i}{m_j} - \frac{\hat{m}_i}{\hat{m}_j} \right)^2}$.

Similarly we define the aggregated measure of closeness of ratios $\frac{l_i}{l_j}$ to ratios $\frac{\hat{l}_i}{\hat{l}_j}$ as $\sigma_l = \frac{1}{n(n-1)} \sqrt{\sum_{i=1}^{n} \sum_{j \neq i=1}^{n} \left(\frac{l_i}{l_j} - \frac{\hat{l}_i}{\hat{l}_j} \right)^2}$.

The resulting σ_m and σ_l are presented in Table 2. It is seen that in all examples a new method provides substantially less values of σ_m and σ_l than the known methods. The best results are obtained with the use of expression (18).

It is worthy to note that in all examples, the condition 1 (see Section 1) for resulting normalized interval weights is verified and the condition 2 is fulfilled in most cases. In our opinion, the condition 2 seems to be rather artificial one, since in many cases it is in contradiction with the more natural desirable property $\sum_{i=1}^{n} [\hat{w}_i] =$ "$near\,1$".

4 Conclusion

A new method for normalization of interval weights based on the so-called "interval extended zero" method is developed. The three desirable properties of normalization procedure are defined. The fist of them is based on the assumption that the sum of normalized interval weights should be an interval centered around 1 with a minimal width. The other desirable property is the remaining of ratios between the means of normalized intervals as close as possible to those of initial interval weights. The third desirable property is based on the intuitively obvious assumption that after a proper normalizing procedure the ratios between interval lengths should be close enough to those before normalization.

With use of six numerical examples, it is shown that a new method provides the results with the properties that are substantially close to the defined three desirable ones than the known methods for normalization of interval weights.

References

1. Bonder, C.G.E., Graan, J.G., Lootsma, F.A.: Multi-criteria decision analysis with fuzzy pairwise comparisons. Fuzzy Sets and Systems 29, 133–143 (1989)
2. Chang, P.T., Lee, E.S.: The estimation of normalized fuzzy weights. Comput. Math. Appl. 29(5), 21–42 (1995)
3. Cleary, J.C.: Logical Arithmetic. Future Computing Systems 2, 125–149 (1987)
4. Denoeux, T.: Reasoning with imprecise belief structures. Internat. J. Approx. Reason. 20, 79–111 (1999)
5. Dong, W.M., Wong, F.S.: Fuzzy weighted averages and implementation of the extension principle. Fuzzy Sets and Systems 21, 183–199 (1987)
6. Dubois, D., Prade, H.: The use of fuzzy numbers in decision analysis. In: Gupta, M.M., Sanchez, E. (eds.) Fuzzy Information and Decision Processes, pp. 309–321. North-Holland, Amsterdam (1982)
7. Guh, Y.Y., Hon, C.C., Wang, K.M., Lee, E.S.: Fuzzy weighted average: a max-min paired elimination method. Comput. Math. Appl. 32(8), 115–123 (1996)
8. Guh, Y.Y., Hon, C.C., Lee, E.S.: Fuzzy weighted average: the linear programming approach via Charnes and Cooper's rule. Fuzzy Sets and Systems 117, 157–160 (2001)
9. Jaulin, L., Kieffir, M., Didrit, O., Walter, E.: Applied Interval Analysis. Springer, London (2001)
10. Kao, C., Liu, S.T.: Fractional programming approach to fuzzy weighted average. Fuzzy Sets and Systems 120, 435–444 (2001)
11. Liou, T.S., Wang, M.J.: Fuzzy weighted average: An improved algorithm. Fuzzy Sets and Systems 49, 307–315 (1992)
12. Mikhailov, L.: Deriving priorities from fuzzy pairwise comparison judgments. Fuzzy Sets and Systems 134, 365–385 (2003)
13. Moore, R.E.: Interval analysis. Prentice-Hall, Englewood Cliffs (1966)
14. Sevastjanov, P., Dymova, L.: Fuzzy solution of interval linear equations. In: Proc. of 7th Int. Conf. Paralel Processing and Applied Mathematics, Gdansk, pp. 1392–1399 (2007)
15. Sevastjanov, P., Dymova, L.: A new method for solving interval and fuzzy equations: linear case. Information Sciences 17, 925–937 (2009)
16. Wang, Y.M., Yang, J.B., Xu, D.L.: Interval weight generation approaches based on consistency test and interval comparison matrices. Appl. Math. Comput. 167, 252–273 (2005)
17. Wang, Y.M., Yang, J.B., Xu, D.L., Chin, K.S.: The evidential reasoning approach for multiple attribute decision making using interval belief degrees. European J. Oper. Res. (in Press)
18. Wang, Y.-M., Elhag, T.M.S.: On the normalization of interval and fuzzy weights. Fuzzy Sets and Systems 157, 2456–2471 (2006)
19. Xu, R.: Fuzzy least-squares priority method in the analytic hierarchy process. Fuzzy Sets and Systems 112, 359–404 (2000)
20. Yager, R.R.: Dempster-Shafer belief structures with interval valued focal weights. Internat. J. Intelligent Systems 16, 497–512 (2001)

A Global Optimization Method for Solving Parametric Linear Systems Whose Input Data Are Rational Functions of Interval Parameters

Iwona Skalna

AGH University of Science and Technology
Faculty of Management, Department of Applied Computer Science
ul. Gramatyka 10, 60–067 Krakow, Poland
skalna@galaxy.uci.agh.edu.pl

Abstract. An interval global optimization method combined with the Direct Method for solving parametric linear systems is used for computing a tight enclosure for the solution set of parametric linear system whose input data are non-linear functions of interval parameters. Revised affine arithmetic is used to handle the nonlinear dependencies. The Direct Method performs the monotonicity test to speed up the convergence of the global optimization. It is shown that the monotonicity test significantly increases the convergence of the global optimization method. Some illustrative examples are solved by the discussed method, and the results are compared to literature data produces by other methods.

Keywords: parametric linear systems, non-affine dependencies, revised affine arithmetic (RAA), global optimization.

1 Introduction

Many real-life problems can be modelled by systems of linear equations or safely transformed to the linear case. When uncertain model parameters are introduced by intervals, then a parametric interval linear system must be appropriately solved to meet all possible scenarios. The problem of solving parametric interval linear system in not trivial. Usually, a parametric solution set is not an interval vector. Hence, instead of the parametric solution set itself, an interval vector containing the parametric solution set, *outer interval solution*, is calculated. The tightest outer interval solution is called the *interval hull solution*. In general case, the problem of computing the hull solution is NP-hard ([27]). However, when the parametric solution is monotone with respect to all interval parameters, interval hull solution can be calculated by solving at most $2n$ real linear systems. The monotonicity approach and the combinatorial approach are favoured by many authors. However, these approaches are valid only under some assumptions. In general case, when the parametric solution is not monotone, the problem of computing the interval hull solution can be solved using optimization techniques. To compute the hull, $2n$ constrained optimization problems must be solved.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 475–484, 2010.

Depending on the problem to be solved, parametric systems can be classified into two types: systems involving affine-linear dependencies and systems involving nonlinear dependencies. So far, mainly problems involving affine-linear dependencies have been solved. In this work, the global optimization method is used to solve parametric linear system involving non-affine dependencies. Nonlinear functions of parameters are transformed into revised affine forms [33] using affine arithmetic. Then, the Direct Method for solving parametric linear systems with affine dependencies is used to calculate inclusion function of the objective function and to perform the monotonicity test to speed up the convergence of the optimization. The Direct Method is an alternative to the more popular self-validated parametric iteration developed by Rump [28] and usually gives better results. The global optimization method produces hull solution of the parametric interval system with affine dependencies which is an outer solution to the original parametric linear system with nonlinear dependencies. The evolutionary optimization method [30] which approximates from below hull solution is used to measure the quality of the solution produced by the proposed approach.

The paper is organized as follows: Sect.2 contains preliminaries on parametric interval linear systems. Section 3 introduces affine arithmetic. Then, revised affine forms and operations on them a re described. Section 4 is devoted to the problem of solving parametric linear systems involving nonlinear dependencies. In Sect.4 the optimization problem is outlined. This is followed by description of the global optimization algorithm. Next, some illustrative examples and the results of computational experiments are presented. The paper ends with summary conclusions.

2 Preliminaries

Real quantities will be denoted by italic faces, while their interval counterparts will be denoted by bold italic faces. Capital letters will be used to denote sets and matrices. Let $\mathbb{IR}$ denote the set of real compact intervals. Then, $\mathbb{IR}^n$ will denote interval vectors, and $\mathbb{IR}^{n\times n}$ will denote square interval matrices ([21]). For an interval $\boldsymbol{x} = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leqslant x \leqslant \overline{x}\}$, some characteristics are defined: the midpoint $\check{x} = (\underline{x} + \overline{x})/2$ and the radius $\mathrm{rad}\,(\boldsymbol{x}) = (\overline{x} - \underline{x})/2$. They are applied to interval vectors and matrices componentwise.

Now, consider linear algebraic system

$$A(p)x(p) = b(p) \ , \tag{1}$$

where $p = (p_1, \ldots, p_k)^T$ is a k-dimensional vector of parameters, $A(p)$ is an $n \times n$ matrix, and $b(p)$ is an n-dimensional vector. The elements of $A(p)$ and $b(p)$ are continuous (nonlinear) functions of parameters:

$$a_{ij}(p) = f_{ij}(p), \qquad b_i(p) = f_i(p) \ , \tag{2}$$

$f_i,\ f_{ij} : \mathbb{R}^k \longrightarrow \mathbb{R},\ i,\ j = 1, \ldots, n.$

When the parameters are considered to be unknown (or uncertain) and vary within prescribed intervals $p_i \in \boldsymbol{p}_i$, $i = 1, \ldots, k$, a family of linear systems is obtained. It is usually written in a compact form

$$A(\boldsymbol{p})x(\boldsymbol{p}) = b(\boldsymbol{p}) \tag{3}$$

and called a *parametric interval linear system*. A set of all solutions, *parametric* (*united*) *solution set*, is defined ([10,12,28]) as follows:

$$S(\boldsymbol{p}) = \{x \mid A(p)x = b(p), \text{ for some } p \in \boldsymbol{p}\} \quad . \tag{4}$$

The solution set $S(\boldsymbol{p})$ is bounded if $A(p)$ is non-singular for every $p \in \boldsymbol{p}$. For a nonempty bounded set $S \subset \mathbb{R}^n$ an *interval hull* is defined by

$$\Box S = [\inf \boldsymbol{S}, \sup \boldsymbol{S}] = \bigcap \{\boldsymbol{Y} \in \mathbb{IR}^n \mid \boldsymbol{S} \subseteq \boldsymbol{Y}\} \quad . \tag{5}$$

3 Affine Arithmetic

Affine arithmetic is an extension of interval arithmetic which enables to take into account dependencies between the variables. In affine arithmetic, a partially unknown quantity x is represented by an affine form $\hat{x}$ which is a first degree polynomial [32]:

$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n \quad , \tag{6}$$

where $x_i \in \mathbb{R}$, $\varepsilon_i \in [-1, 1]$. The central value of the affine form $\hat{x}$ is denoted by x_0, x_i are partial deviations, and ε_i are dummy variables [35] (noise symbols [32], variation variables [6]). The magnitude of the dependency between two quantities x and y represented by affine forms $\hat{x}$ and $\hat{y}$ is determined by partial deviations x_i and y_i. The sign of the partial deviations defines the direction of the correlation between x and y.

The *fundamental invariant of affine arithmetic* states that, at any instant between AA operations, there is a single assignment of values from $[-1, 1]$ to each of the dummy variables in use that makes the value of every affine form $\hat{x}$ equal to the true value of the corresponding ideal quantity x.

3.1 Revised Affine Arithmetic

One of the limitations of the standard affine arithmetic is that the number of dummy variables grows gradually during the computation, and the computation cost heavily depends on this number. The remedy, following [11], [15], [33], is to use revised affine forms. The revised affine form, defined as

$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n + e_x[-1, 1] \quad , \tag{7}$$

consists of two separate parts: the standard affine part of length n, and the error $[-e_x, e_x]$, $e_x > 0$ (e_x is called an error variable). In rigorous computations, e_x is also used to accumulate the rounding errors in floating-point arithmetic.

3.2 Affine Operations

In order to evaluate a formula with RAA, each elementary operation on real quantities must be replaced by a corresponding operation on their affine forms, returning an affine form.

Let

$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n + e_x[-1,\, 1] \ ,$$
$$\hat{y} = y_0 + y_1\varepsilon_1 + \ldots + y_n\varepsilon_n + e_y[-1,\, 1] \ .$$

If an RAA operation $f(\hat{x},\, \hat{y})$ is an affine function of $\hat{x}$ and $\hat{y}$, then $\hat{z} = f(\hat{x},\, \hat{y})$ is an affine combination of the dummy variables ε_i, and the accumulative error e_z is a combination of error variables e_x, e_y. Thus, for any α, β, $\gamma \in \mathbb{R}$,

$$\hat{z} = \alpha\hat{x} + \beta\hat{y} + \gamma = (\alpha x_0 + \beta y_0 + \gamma) + \sum_{i=1}^{n}(\alpha x_i + \beta y_i)\varepsilon_i + (|\alpha|e_x + |\beta|e_y)[-1,\, 1] \ .$$

In particular, $\hat{x} - \hat{x} = 0$.

Now, consider multiplication of affine forms $\hat{z} = f(\hat{x},\, \hat{y}) = \hat{x}\hat{y}$. The product is a quadratic polynomial $f^*(\varepsilon_1,\, \ldots,\, \varepsilon_n,\, e_x,\, e_y)$ of the dummy variables and error variables. Following [32], the range estimate $\pm\mathrm{rad}(\hat{x})\mathrm{rad}(\hat{y})$ for the nonlinear term might be used. Then, the multiplication will return

$$\hat{z} = x_0y_0 + \sum_{i=1}^{n}(x_0y_i + x_iy_0)\varepsilon_i + (|x_0|e_y + |y_0|e_x + \mathrm{rad}(\hat{x})\mathrm{rad}(\hat{y}))[-1,\, 1] \ .$$

Based on the multiplication formula proposed by Kolev [11], another multiplication method have been suggested in [33]:

$$\begin{aligned}\hat{x}\cdot\hat{y} = &\left(x_0y_0 + 0.5\sum_{i=1}^{n}x_iy_i\right) + \sum_{i=1}^{n}(x_0y_i + x_iy_0)\varepsilon_i \\ &+ \left(e_xe_y + e_yu + e_xv + uv - 0.5\sum_{i=1}^{n}|x_iy_i|\right)[-1,\, 1] \ ,\end{aligned} \tag{8}$$

where $u = |x_1| + |x_2| + \ldots + |x_n|$, $v = |y_1| + |y_2| + \ldots + |y_n|$.

The reciprocal $1/\hat{y}$ of an affine form $\hat{y} = y_0 + y_1\varepsilon_1 + y_2\varepsilon_2 + \ldots + y_n\varepsilon_n + e_y[-1,\, 1]$ can be calculated using Chebyshev approximation ([11], [18]). Then, the division rule can be defined as (cf. [7]):

$$\frac{\hat{x}}{\hat{y}} = \frac{x_0}{y_0} + \frac{1}{\hat{y}}\left(\sum_{i=1}^{n}\left(x_i - \frac{x_0}{y_0}y_i\right)\varepsilon_i + |e_x - \frac{x_0}{y_0}e_y|[-1,\, 1]\right). \tag{9}$$

In particular, $\hat{x}/\hat{x} = 1$.

For elementary univariate functions, an affine approximations can be found via Theorem 2 from [32] or Theorem 1 from [33]. For the purposes of this paper, Chebyshev approximation is used.

4 Solving Parametric Linear Systems With Non-affine Dependencies

Consider system of parametric linear interval equations (3). The coefficients are then non-affine function of interval parameters $a_{ij}(\boldsymbol{p}) = f_{ij}(\boldsymbol{p})$, $b_i(\boldsymbol{p}) = f_i(\boldsymbol{p})$. Affine arithmetic enables to transform the nonlinear coefficients into revised affine forms (7). A new linear system $A(\boldsymbol{q})x(\boldsymbol{q}) = b(\boldsymbol{q})$ with affine-linear dependencies:

$$\begin{aligned} a_{ij}(\boldsymbol{q}) &= \omega_{ij0} + \omega_{ij}^T \boldsymbol{q} \ , \\ b_i(\boldsymbol{q}) &= \omega_{i0} + \omega_i^T \boldsymbol{q} \ , \end{aligned} \tag{10}$$

where ω_{ij}, $\omega_i \in \mathbb{R}^l$ $(i, j = 1, \ldots, n)$, $\boldsymbol{q}_i = [-1, 1]$, is thus obtained.

A parametric linear system with coefficients given by formula (10) can be solved using the Direct Method [29] for solving parametric linear systems. Affine forms (10) are inclusion functions of non-affine coefficients (2). The Direct Method will then produce an interval enclosure of the solution set of the original parametric linear system with non-affine dependencies.

It is worth to mention that the iterative method proposed by Rump [28] is widely used to solve parametric linear systems with interval coefficients, while the Direct Method usually gives better results for problems involving affine-linear dependencies.

5 Optimization Problem

The problem of computing the interval hull solution for the parametric linear system (3) with affine dependencies (10) can be written as a problem of solving $2n$ (n is the number of equations) constrained optimization problems:

$$\min_{q \in ([-1,\,1])^l} x_i(q), \qquad \max_{q \in ([-1,\,1])^l} x_i(q) \ , \tag{11}$$

$(i = 1, \ldots, n)$, where $x_i(q) = \left\{A(q)^{-1} b(q)\right\}_i$ is an implicitly given objective function for the i-th optimization problem.

Theorem 1. *Let $A(\boldsymbol{q})$ be regular, $\boldsymbol{q} \in \mathbb{IR}^l$. Let $x^i_{\min}$ and $x^i_{\max}$ denote global solutions of the i-th minimization and maximization problems (11), respectively. Then the interval vector*

$$\boldsymbol{x} = [x_{\min},\, x_{\max}] = \left(\left[x^i_{\min},\, x^i_{\max}\right]\right)_{i=1}^{n} = \square S(\boldsymbol{e}) \ . \tag{12}$$

5.1 Global Optimization

Optimization problems (11) are solved using an interval global optimization (GOM) method with selected acceleration techniques. The maximization problem is transformed into the respective minimization problem, hence the research focuses on global minimization. Inclusion functions of the objective functions $x_i(p)$ are calculated using the Direct Method.

5.2 Monotonicity Test

The monotonicity test is used to figure out whether the function f is strictly monotone in a subbox $\boldsymbol{y} \subseteq \boldsymbol{x}$. Then, $\boldsymbol{y}$ cannot contain a global minimizer in its interior. Therefore, if f satisfies

$$\frac{\partial f}{\partial x_i}(\boldsymbol{y}) < 0 \quad \vee \quad \frac{\partial f}{\partial x_i}(\boldsymbol{y}) > 0 \ , \tag{13}$$

then the subbox $\boldsymbol{y}$ can be reduced to one of its edges. The proper endpoint, degenerated subbox, is then entered into the list.

Monotonicity test is performed this time using the MCM [31] method which is based on the Direct Method ([29]) for solving parametric linear systems. This special treatment is necessary since the objective functions are given implicitly (see Sect.5). The detailed description of the method for checking the monotonicity can be found in [31].

5.3 Global Optimization Algorithm

Below the global optimization algorithm with the monotonicity test is presented.

1. Set $\boldsymbol{y} = \boldsymbol{q}$, $f = \min x(\boldsymbol{y})$
2. Initialize the list $L = \{(f, \boldsymbol{y})\}$ and the cutoff level $z = \max x(\boldsymbol{y})$
3. Remove $(f, \boldsymbol{y})$ from the list L
4. Bisect $\boldsymbol{y}$: $\boldsymbol{y}_1 \cup \boldsymbol{y}_2$
5. Calculate $x(\boldsymbol{y}_i)$, $f_i = \min x(\boldsymbol{y}_i)$ $(i = 1, 2)$, $z = \min\limits_i \{z, \max x(\boldsymbol{y}_i)\}$
6. Discard the pair $(f_i, \boldsymbol{y}_i)$ if $f_i > z$ $(i = 1, 2)$ (cutoff test)
7. Perform the monotonicity test using the MCM method
8. Add any remaining pairs to L so that the list remains increasingly sorted by function value; if the list becomes empty then **STOP**
9. If $\mathrm{w}(\boldsymbol{y}) \leq \varepsilon$ (where ε is an algorithm parameter), then print $x(\boldsymbol{y})$, $\boldsymbol{y}$ and **STOP**, else **GOTO** 3.

6 Examples and Numerical Results

To check the performance of the proposed approach, some illustrative examples of parametric interval linear systems with general (non-affine) dependencies ([5,13,23]) are considered. The Evolutionary Optimization Method (EOM) (cf. [30]) is used to verify the overestimation. The EOM method produces a very good inner approximation of the hull solution of the original parametric linear systems with nonlinear dependencies.

Example 1 ([5], [13]).

$$\begin{pmatrix} -(p_1 + p_2)p_4 & p_2 p_4 \\ p_5 & p_3 p_5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix},$$

$p_1, p_3 \in [0.96, 0.98]$, $p_2 \in [1.92, 1.96]$, $p_4, p_5 \in [0.48, 0.5]$.

Table 1. Solutions of Ex.1: the solution enclosures generated by the Global Optimization Method and the Evolutionary Optimization Method

	GOM	EOM
$\boldsymbol{x}_1$	[0.3773261, 0.4545408]	[0.3776424, 0.4541765]
$\boldsymbol{x}_2$	[1.6259488, 1.7280411]	[1.6260162, 1.7272530]

Example 2 ([5], [13])

$$\begin{pmatrix} -(p_1+1)p_2 & p_1p_3 & p_2 \\ p_2p_4 & p_2^2 & 1 \\ p_1p_2 & p_3p_5 & \sqrt{p_2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix},$$

$p_1 \in [1, 1.2]$, $p_2 \in [2, 2.2]$, $p_3 \in [0.5, 0.51]$, p_4, $p_5 \in [0.39, 0.4]$.

Table 2. Solutions of Ex.2: the solution enclosures generated by the Global Optimization Method and the Evolutionary Optimization Method

	GOM	EOM
$\boldsymbol{x}_1$	[0.0563815, 0.0652607]	[0.0567702, 0.0652458]
$\boldsymbol{x}_2$	[0.0775223, 0.0899316]	[0.0775458, 0.0894860]
$\boldsymbol{x}_3$	[0.5594312, 0.6060671]	[0.5597984, 0.6054807]

Example 3 ([3], [24]) Consider parametric linear system described by the following relations (all components not specified below are equal to zero):

$$\begin{aligned}
&a_{11} = a_{66} = u_7 + 12u_6, \quad a_{13} = a_{31} = a_{68} = a_{86} = 6u_4,\\
&a_{16} = a_{61} = -u_7, \quad a_{22} = a_{77} = u_8 + 12u_5,\\
&a_{24} = a_{25} = a_{42} = -a_{47} = a_{52} = -a_{57} = -a_{74} = -a_{75} = -a_{78} = -a_{87} = 6u_3,\\
&a_{27} = a_{72} = -12u_5, \quad a_{33} = a_{55} = a_{88} = \alpha + 4u_2,\\
&a_{34} = a_{43} = a_{58} = a_{85} = -\alpha, \quad a_{44} = \alpha + 4u_1, \quad a_{45} = a_{54} = 2u_3,\\
&b_1 = H \ ,
\end{aligned}$$

where

$$\begin{aligned}
u_1 &= (E_bI_b)/L_b, & u_2 &= (E_cI_c)/L_c,\\
u_3 &= (E_bI_b)/L_b^2, & u_4 &= (E_cI_c)/L_c^2,\\
u_5 &= (E_bI_b)/L_b^3, & u_6 &= (E_cI_c)/L_c^3,\\
u_7 &= (A_bE_b)/L_b, & u_8 &= (A_cE_c)/L_c,
\end{aligned}$$

Typical nominal parameter values E_b, E_c, I_b, I_c, A_b, A_c, α, H, and the corresponding worst case uncertainties as proposed in [3] are shown in Tab.3. Linear system, where L_b, L_c and H are replaced by their nominal values, is solved with parameter uncertainties which are 1% of the value in the last column of Tab.3:

E_b, $E_c \in [28965200, 29034800]$, $I_b \in [509.49, 510.51]$, $I_c \in [271.728, 272.272]$
$A_b \in [10.287, 10.313]$, $A_c \in [14, 3856, 14.4144]$, $\alpha \in [276195960, 278726040]$.

Table 3. Parameters for Ex.3, their nominal values and the worst case uncertainties

Parameter	Nominal value	Uncertainty
E_b	$29 \cdot 10^6$ lbs/in^2	$\pm 348 \cdot 10^4$
E_c	$29 \cdot 10^6$ lbs/in^2	$\pm 348 \cdot 10^4$
I_b	510 in^4	± 51
I_c	272 in^4	± 27.2
A_b	10.3 in^2	± 1.3
A_c	14.4 in^2	± 1.44
α	$2.77461 \cdot 10^9$ lb-in/rad	$\pm 1.26504 \cdot 10^9$
H	5305.5 lbs	
L_b	144 in	
L_c	288 in	

Table 4 lists both the guaranteed outer enclosure of the parametric solution set produced by GOM method and inner estimation of the hull solution obtained by EOM method. Coincident digits are underlined.

Table 4. Solutions of Ex.3: the solution enclosures generated by the Global Optimization Method and the Evolutionary Optimization Method

	GOM	EOM
$\boldsymbol{x}_1$	[0.1528387, 0.1536945]	[0.1528665, 0.1536661]
$\boldsymbol{x}_2$	[0.0003229, 0.0003262]	[0.0003229, 0.0003261]
$\boldsymbol{x}_3$	[−0.0009678, −0.0009615]	[−0.0009676, −0.0009617]
$\boldsymbol{x}_4$	[−0.0004672, −0.0004641]	[−0.0004671, −0.0004642]
$\boldsymbol{x}_5$	[−0.0004285, −0.0004255]	[−0.0004284, −0.0004256]
$\boldsymbol{x}_6$	[0.1502880, 0.1511376]	[0.1503158, 0.1511092]
$\boldsymbol{x}_7$	[−0.0006700, −0.0006625]	[−0.0006699, −0.0006626]
$\boldsymbol{x}_8$	[−0.0009359, −0.0009296]	[−0.0009357, −0.0009298]

7 Conclusions

The global optimization method for solving parametric linear systems involving nonlinear dependencies is proposed. It is demonstrated that guaranteed sharp solution enclosures are generated by the proposed method. Contrary to other approaches, present method is highly automated. It requires no preliminary specialized construction method and no additional tools. The quality of the solution depends only on a quality of approximation of non-affine forms by revised affine forms. The main drawback of the proposed approach is the computational complexity. However, this can be overcome by using parallel programming which will be the subject of the future work.

References

1. Aughenbaugh, J., Paredis, C.: Why are intervals and imprecisions important in engineering design? In: Muhanna, R.L., Mullen, R.L. (eds.) Proceedings of the NSF Workshop on Reliable Engineering Computing (REC 2006), pp. 319–340 (2006)
2. Casado, L.G., Garcia, I., Csendes, T.: A new multisection technique in interval methods for global optimization. Computing 65, 263–269 (2000)
3. Corliss, G., Foley, C., Kearfott, R.B.: Formulation for reliable analysis of structural frames. Reliable Computing 13, 125–147 (2007)
4. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
5. El-Owny, H.: Parametric Linear System of Equations, whose Elements are Nonlinear Functions. In: 12th GAMM - IMACS International Symposion on Scientific Computing, Computer Arithmetic and Validated Numerics, SCAN (2006)
6. Fang, C.F., Chen, T., Rutenbar, R.: Floating-point error analysis based on affine arithmetic. In: Proceedings of IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP), vol. 2, pp. 561–564 (2003)
7. Hansen, E.R.: A Generalised Interval Arithmetic. LNCS, vol. 29, pp. 7–18. Springer, Heidelberg (1975)
8. Hansen, E.: Global optimization using interval analysis - the multidimensional case. Numerical Mathematics 43, 247–270 (1980)
9. Hansen, E.: Global optimization using interval analysis. Marcel Dekker, New York (1992)
10. Jansson, C.: Interval linear systems with symmetric matrices, skew-symmetric matrices and dependencies in the right hand side. Computing 46(3), 265–274 (1991)
11. Kolev, L.V.: An Improved Interval Linearization for Solving Nonlinear Problems. Numerical Algorithms 37(1-4), 213–224 (2004)
12. Kolev, L.V.: A method for outer interval solution of linear parametric systems. Reliable Computing 10(3), 227–239 (2004)
13. Kolev, L.V.: Solving Linear Systems whose Elements are Non-linear Functions of Intervals. Numerical Algorithms 37, 213–224 (2004)
14. Lallemand, B., Plessis, G., Tison, T., Level, P.: Modal Behaviour of Structures Defined by Imprecise Geometric Parameters. Proceedings of International Modal Analysis Cconference 4062(2), 1422–1428 (2000)
15. Messine, F.: Extentions of Affine Arithmetic: Application to Unconstrained Global Optimization. Journal of Universal Computer Science 8(11), 992–1015 (2002)
16. Mullen, R., Muhanna, R.L.: Efficient interval methods for finite element solutions. In Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications, 161–168 (2002)
17. Miyajima, S., Miyata, T., Kashiwagi, M.: On the Best Multiplication of the Affine Arithmetic. Transactions of the Institute of Electronics, Information and Communication Engineers J86-A(2), 150–159 (2003)
18. Miyajima, S., Kashiwagi, M.: A dividing method utilizing the best multiplication in affine arithmetic. IEICE Electronics Express 1(7), 176–181 (2004)
19. Muhanna, R.L., Erdolen, A.: Geometric uncertainty in truss systems: an interval approach. In: Muhanna, R.L., Mullen, R.L. (eds.) Proceedings of the NSF Workshop on Reliable Engineering Computing (REC), pp. 229–244 (2006)
20. Muhanna, R.L., Kreinovich, V., Solin, P., Cheesa, J., Araiza, R., Xiang, G.: Interval finite element method: New directions. In: Muhanna, R.L., Mullen, R.L. (eds.) Proceedings of the NSF Workshop on Reliable Engineering Computing (REC 2006), Savannah, Georgia, USA, pp. 229–244 (2006)

21. Neumaier, A.: Interval Methods for Systems of Equations, Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge (1990)
22. Neumaier, A., Pownuk, A.: Linear Systems with Large Uncertainties, with Applications to Truss Structures. Reliable Computing 13(2), 149–172 (2007)
23. Popova, E.D.: Solving Linear Systems Whose Input Data Are Rational Functions of Interval Parameters. Numerical Methods and Applications, 345–352 (2006)
24. Popova, E., Iankov, R., Bonev, Z.: Bounding the Response of Mechanical Structures with Uncertainties in all the Parameters. In: Muhanna, R.L., Mullen, R.L. (eds.) Proceedings of the NSF Workshop on Reliable Engineering Computing (REC), pp. 245–265 (2006)
25. Ratz, D.: Automatische Ergebnisverifikation bei globalen Optimierungsproblemen. Ph.D. Thesis, Universität Karlsruhe, Karlsruhe, Germany (1992)
26. Ratz, D., Csendes, T.: Subdivision Direction Selection in Interval Methods for Global Optimization. SIAM Journal on Numerical Analysis 34(3), 922–938 (1997)
27. Rohn, J., Kreinovich, V.: Computing exact componentwise bounds on solutions of linear systems with interval data is NP-hard. SIAM Journal on Matrix Analysis and Applications (SIMAX) 16, 415–420 (1995)
28. Rump, S.M.: Verification Methods for Dense and Sparse Systems of Equations. In: Herzberger, J. (ed.) Topics in Validated Computations – Studies in Computational Mathematics, pp. 63–136. Elsevier, Amsterdam (1994)
29. Skalna, I.: A Method for Outer Interval Solution of Systems of Linear Equations Depending Linearly on Interval Parameters. Reliable Computing 12(2), 107–120 (2006)
30. Skalna, I.: Evolutionary Optimization Method for Approximating the Solution Set Hull of Parametric Linear Systems. In: Boyanov, T., Dimova, S., Georgiev, K., Nikolov, G. (eds.) NMA 2006. LNCS, vol. 4310, pp. 361–368. Springer, Heidelberg (2007)
31. Skalna, I.: On Checking the Monotonicity of Parametric Interval Solution. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 1400–1409. Springer, Heidelberg (2008)
32. de Figueiredo, L.H., Stolfi, J.: Self-Validated Numerical Methods and Applications. Brazilian Mathematics Colloquium monograph (1997)
33. Vu, X., Sam-Haroud, D., Faltings, B.V.: A Generic Scheme for Combining Multiple Inclusion Representations in Numerical Constraint Propagation. Artificial Intelligence Laboratory, Swiss Federal Institute of Technology in Lausanne (EPFL), Tech. Report No. IC/2004/39 (2004)
34. Zalewski, B., Muhanna, R.L., Mullen, R.: Bounding the response of mechanical structures with uncertainties in all the parameters. In: Muhannah, R.L., Mullen, R.L. (eds.) Proceedings of the NSF Workshop on Reliable Engineering Computing (REC 2006), pp. 439–456 (2006)
35. http://www.ic.unicamp.br/ stolfi/EXPORT/projects/affine-arith/Welcome.html

Direct Method for Solving Parametric Interval Linear Systems with Non-affine Dependencies

Iwona Skalna

AGH University of Science and Technology
Faculty of Management, Department of Applied Computer Science
ul. Gramatyka 10, 60–067 Krakow, Poland
skalna@galaxy.uci.agh.edu.pl

Abstract. Many real-life problems can be modelled by systems of linear equations or safely transformed to the linear case. When uncertain model parameters are introduced by intervals, then a parametric interval linear system must properly be solved to meet all possible scenarios and yield useful results. In general case, system coefficients are nonlinear functions of parameters. The Direct Method for solving such systems is proposed. Affine arithmetic is used to handle nonlinear dependencies. Some illustrative examples are solved and the results are compared to the literature data produced by other methods.

Keywords: parametric linear systems, non-affine dependencies, affine arithmetic (AA), Direct Method.

1 Introduction

When dealing with real-world problems, one can rarely avoid uncertainty. At the empirical level, uncertainty is an inseparable companion of almost any measurement. The difference between the measured and the actual values is called a *measurement error*. Since the absolute value of the measurement error can usually be bounded, it is therefore guaranteed that the actual (unknown) value of the desired quantity belongs to the interval with the midpoint being the measured value, and the radius being the upper bound for the (absolute) value of the possible errors. Moreover, outward rounding forces the result to be the interval approximation of the correct real interval.

One of the main obstacles in the wide-spread use of interval methods is that the range estimates computed with naive (straightforward) interval arithmetic (IA) tend to be too large, especially in complicated expressions or long iterative computations. This is mainly due to the *dependency problem* (*interval dependency*). The formulas for the basic arithmetic operations assume that the (unknown) values of the operands vary independently over their given intervals. Therefore, when arguments of expression are partially dependent on each other or expression contains multiple instances of one or more variables, the result obtained by interval arithmetic may be much wider then the exact range of the result quantity. To decrease excess width, it is therefore desirable to keep

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 485–494, 2010.

track of how the values in the expression depend on each other and how the corresponding intermediate result depend on the input. This idea has been successfully implemented by two approaches, Hansen's generalized interval arithmetic [5] and affine arithmetic [29]. Hansen's generalized interval arithmetic and affine arithmetic keep track of correlations between computed and input quantities, and therefore are able to provide much tighter bounds for the computed quantities. Affine arithmetic is somewhat similar to Hansen's generalized interval arithmetic, but differs in several important details (cf. [20]). There are also other models that provide first-order approximations of dependencies in interval expressions, including centered forms [18], first-order Taylor arithmetic [18], and the ellipsoidal calculus of Chernousko, Kurzhanski, and Ovseevich [14]. Affine arithmetic seems to have several advantages over them, including a wider range of applications and a more convenient programming interface (cf. [29]).

Many real-life problems can be described by systems of linear equations, or safely transformed to the linear case, involving uncertain model parameters. When uncertain parameters are introduced by intervals, then a *parametric interval linear system* is obtained with components that are, in general case, nonlinear functions of parameters. Several methods for solving parametric interval linear systems with affine linear dependencies has been developed during last years [1],[3], [19], [22], [24], [26],[28]. The Direct Method for solving such systems been proposed in [30].

The goal is to expand the Direct Method to solve parametric linear systems with general (nonlinear) dependencies. Affine arithmetic is used to estimate nonlinear functions of parameters by affine linear forms. This results in a new parametric linear system with affine-linear dependencies which can be solved using the Direct Method.

The paper is organised in five sections. In Sect.2, preliminary information on parametric interval linear systems is presented. Section 3 introduces affine arithmetic. The Direct Method for solving parametric interval linear systems with non-affine dependencies is described in Sect.4. This is followed by some numerical examples. The paper ends with summary conclusions.

2 Parametric Interval Linear Systems

The set of n-dimensional interval vectors and the set of $n \times m$ interval matrices will be denoted, respectively, by $\mathbb{IR}^n$ and $\mathbb{IR}^{n\times m}$. Italic faces will be used for real quantities, while bold italic faces will denote their interval counterparts.

For a square interval matrix $\boldsymbol{A} = (\boldsymbol{a}_{ij})$ the *comparison matrix* $\langle \boldsymbol{A} \rangle$ is defined as the matrix with diagonal components $\langle \boldsymbol{a} \rangle_{ii} = \langle \boldsymbol{a}_{ii} \rangle$ and off-diagonal components $\langle \boldsymbol{a} \rangle_{ij} = -|\boldsymbol{a}_{ij}|$ (for $i \neq j$), where $\langle \boldsymbol{a}_{ij} \rangle = \min\{|a_{ij}| \mid a_{ij} \in \boldsymbol{a}_{ij}\}$, $|\boldsymbol{a}_{ij}| = \max\{|a_{ij}| \mid a_{ij} \in \boldsymbol{a}_{ij}\}$.

A square interval matrix $\boldsymbol{A}$ is an H-matrix iff there exist a real vector $u > 0$ such that $\langle \boldsymbol{A} \rangle u > 0$.

Now, consider linear algebraic system

$$A(p)x(p) = b(p) \ , \tag{1}$$

where $p = (p_1, \ldots, p_k)^T$ is a k-dimensional vector of parameters, $A(p)$ is an $n \times n$ matrix, and $b(p)$ is an n-dimensional vector. The elements of $A(p)$ and $b(p)$ are continuous (nonlinear) functions of parameters:

$$a_{ij}(p) = f_{ij}(p), \qquad b_i(p) = f_i(p) \ , \tag{2}$$

$f_i, f_{ij} : \mathbb{R}^k \longrightarrow \mathbb{R}$, $i, j = 1, \ldots, n$.

When the parameters are considered to be unknown (or uncertain) and vary within prescribed intervals $p_i \in \boldsymbol{p}_i$, $i = 1, \ldots, k$, a family of linear systems is obtained. It is usually written in a compact form

$$A(\boldsymbol{p})x(\boldsymbol{p}) = b(\boldsymbol{p}) \tag{3}$$

and called a *parametric interval linear system*. The set of all solutions, *parametric* (*united*) *solution set* is defined as follows:

$$S(\boldsymbol{p}) = \{x \mid A(p)x = b(p), \text{ for some } p \in \boldsymbol{p}\} \ . \tag{4}$$

The solution set is bounded if $A(p)$ is non-singular for every $p \in \boldsymbol{p}$. For a nonempty bounded set $S \subset \mathbb{R}^n$ an interval hull is defined as follows:

$$\Box S = \bigcap \{\boldsymbol{Y} \in \mathbb{IR}^n \mid S \subseteq \boldsymbol{Y}\} \ . \tag{5}$$

It is quite expensive to obtain the solution set $S(\boldsymbol{p})$ or its interval hull $\Box S(\boldsymbol{p})$ (*interval hull solution*). In general case, the problem of computing the hull solution is NP-hard. Therefore, an interval vector $\boldsymbol{x}^* \supseteq \Box S(\boldsymbol{p}) \supseteq S(\boldsymbol{p})$, called *outer interval solution*, is computed instead, and the goal is $\boldsymbol{x}^*$ to be as narrow as possible.

3 Affine Arithmetic

In affine arithmetic, a partially unknown quantity x is represented by an affine form $\hat{x}$ which is a first degree polynomial [29]:

$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n \ , \tag{6}$$

where $x_i \in \mathbb{R}$, $\varepsilon_i \in [-1, 1]$. The central value of the affine form $\hat{x}$ is denoted by x_0, x_i are partial deviations, and ε_i are dummy variables [33], also called noise symbols [29] or variation symbols [4]. Affine forms enables to keep track of dependencies between the variables. The magnitude of the dependency between two quantities x and y represented by affine forms $\hat{x}$ and $\hat{y}$ is determined by partial deviations x_i and y_i. The sign of the partial deviations defines the direction of the correlation between x and y.

An affine form $\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n$ implies an interval for the corresponding variable x:

$$x \in \boldsymbol{x} = [x_0 + r, x_0 - r] \ , \tag{7}$$

where r is the sum of partial deviations. This is the smallest interval that contains all possible values of x, assuming that each ε_i ranges independently over the $[-1, 1]$. Conversely, any interval $\boldsymbol{x} = [\check{x}-r, \check{x}+r]$ representing some quantity x has equivalent affine form given by $\hat{x} = \check{x} + r\varepsilon_\nu$, where the dummy variable ε_ν must be distinct from all other dummy variables used so far in computations. For example, consider the following affine forms:

$$\begin{aligned} \hat{x} &= 9 + 1\varepsilon_1 \qquad\quad - 2\varepsilon_3 + 2\varepsilon_4 \ , \\ \hat{y} &= 6 + 2\varepsilon_1 + 1\varepsilon_2 \qquad - 1\varepsilon_4 \ . \end{aligned}$$

They imply intervals $\boldsymbol{x} = [4, 14]$ for $\hat{x}$ and $\boldsymbol{y} = [2, 10]$ for $\hat{y}$. However, since $\hat{x}$ and $\hat{y}$ include the same dummy variables ε_1 and ε_4 they are not entirely independent of each other. In interval representation this information is lost, e.g. $\hat{x}-\hat{y}$ implies $[-4, 10]$, and $\boldsymbol{x} - \boldsymbol{y} = [-6, 12]$.

The *fundamental invariant of affine arithmetic* states that, at any instant between AA operations, there is a single assignment of values from $[-1, 1]$ to each of the dummy variables in use that makes the value of every affine form $\hat{x}$ equal to the true value of the corresponding ideal quantity x.

3.1 Affine Operations

In order to evaluate a formula with AA, each elementary operation on real quantities must be replaced by a corresponding operation on their affine forms, returning an affine form.

Let

$$\begin{aligned} \hat{x} &= x_0 + x_1\varepsilon_1 + \ldots + x_n\varepsilon_n \ , \\ \hat{y} &= y_0 + y_1\varepsilon_1 + \ldots + y_n\varepsilon_n \ . \end{aligned}$$

If an AA operation $f(\hat{x}, \hat{y})$ is an affine function of $\hat{x}$ and $\hat{y}$, then $\hat{z} = f(\hat{x}, \hat{y})$ is an affine combination of the dummy variables ε_i. For any $\alpha, \beta, \gamma \in \mathbb{R}$,

$$\hat{z} = \alpha\hat{x} + \beta\hat{y} + \gamma = (\alpha x_0 + \beta y_0 + \gamma) + (\alpha x_1 + \beta y_1)\varepsilon_1 + \ldots + (\alpha x_n + \beta y_n)\varepsilon_n \ .$$

In particular, $\hat{x} - \hat{x} = 0$.

Now, consider multiplication of affine forms $\hat{z} = f(\hat{x}, \hat{y}) = \hat{x}\hat{y}$. The product of affine forms is a quadratic polynomial $f^*(\varepsilon_1, \ldots, \varepsilon_n)$ of the dummy variables. In general, the problem of computing the (exact) range of quadratic function under interval uncertainty is NP-hard. Stolfi [29] proposes to use the range estimate $\pm\mathrm{rad}\,(\hat{x})\mathrm{rad}\,(\hat{y})$ for the nonlinear term in f^* and return

$$\hat{z} = x_0y_0 + \sum_{i=1}^{n}(x_0y_i + x_iy_0)\varepsilon_i + \mathrm{rad}\,(\hat{x})\mathrm{rad}\,(\hat{y})\varepsilon_{\mathrm{new}} \ .$$

Another multiplication method have been suggested in [7]:

$$\begin{aligned} \hat{x}\cdot\hat{y} = (x_0y_0 + 0.5\sum_{i=1}^{n} x_iy_i) + \sum_{i=1}^{n}(x_0y_i + x_iy_0)\varepsilon_i \\ + \left(\sum_{i=1}^{n}|x_i|\sum_{i=1}^{n}|y_i| - 0.5\sum_{i=1}^{n}|x_iy_i|\right)\varepsilon_{\mathrm{new}} \ . \end{aligned} \tag{8}$$

The method for the best multiplication of affine forms has been proposed in [16]. It gives the narrowest inclusion, which is the theoretical limit of the multiplication, thought it requires much more computation than other methods.

The reciprocal $1/\hat{y}$ of an affine form $\hat{y} = y_0 + y_1\varepsilon_1 + \ldots + y_n\varepsilon_n$ can be calculated using Chebyshev approximation ([7], [17]). Then, the division rule, as suggested in [7] (cf. [5]), can be defined as:

$$\frac{\hat{x}}{\hat{y}} = \hat{x} \cdot \frac{1}{\hat{y}} = \frac{x_0}{y_0} + \frac{1}{\hat{y}} \sum_{i=1}^{n} \left(x_i - \frac{x_0}{y_0} y_i \right) \varepsilon_i \ . \tag{9}$$

In particular, $\hat{x}/\hat{x} = 1$.

For elementary univariate functions, affine approximations can be found via Theorem 2 from [29] or Theorem 1 from [32]. For the purposes of this paper, Chebyshev approximation is used.

For operations other than negation, rounding errors must be taken into account. In order to preserve the fundamental invariant, whenever a computed coefficient differs from its correct value, this error must be accounted by adding an extra term with a dummy variable that does not occur in any other affine form [29], [32].

4 Direct Method

4.1 Background

Consider a system of parametric linear interval equations (3) involving affine-linear dependencies:

$$a_{ij}(\boldsymbol{p}) = \omega_{ij0} + \omega_{ij}^T \boldsymbol{p}, \qquad b_i(\boldsymbol{p}) = \omega_{i0} + \omega_i^T \boldsymbol{p} \ , \tag{10}$$

where $\omega_{i0}, \omega_{ij0} \in \mathbb{R}$, $\omega_{ij}, \omega_i \in \mathbb{R}^k$, $i, j = 1, \ldots, n$. The Direct Method for solving such systems has been proposed in [30]. The method is based on following

Theorem 1. *([30]) Let $A(\boldsymbol{p})x(\boldsymbol{p}) = b(\boldsymbol{p})$ with $\boldsymbol{p} \in \mathrm{IR}^k$, $a_{ij}(\boldsymbol{p})$, $b_i(\boldsymbol{p})$ be given by (10), $R \in \mathbb{R}^{n\times n}$, and $\tilde{x} \in \mathbb{R}^n$. Define $D(\boldsymbol{p})$ and $z(\boldsymbol{p})$ with components*

$$d_{ij}(\boldsymbol{p}) = \sum_{\nu=1}^{n} r_{i\nu}\omega_{\nu j0} + \Big(\sum_{\nu=1}^{n} r_{i\nu}\omega_{\nu j} \Big)^T \boldsymbol{p} \ , \tag{11}$$

$$z_i(\boldsymbol{p}) = \sum_{j=1}^{n} r_{ij} \Big(\omega_{j0} - \sum_{\nu=1}^{n} \omega_{j\nu 0}\tilde{x}_\nu \Big) + \left(\sum_{j=1}^{n} r_{ij} \Big(\omega_j - \sum_{\nu=1}^{n} \omega_{j\nu}\tilde{x}_\nu \Big) \right)^T \boldsymbol{p} \ . \tag{12}$$

If $D(\boldsymbol{p})$ is an H-matrix then

$$\square S(\boldsymbol{p}) \subseteq \tilde{x} + \langle D(\boldsymbol{p}) \rangle^{-1} |z(\boldsymbol{p})| [-1, 1] \ .$$

It is recommended to choose $R = A^{-1}(\check{p})$ and $\tilde{x} = A^{-1}(\check{p}) \cdot b(\check{p})$, where $\check{p}$ is the midpoint vector of $\boldsymbol{p}$, so that $D(\boldsymbol{p})$ and $z(\boldsymbol{p})$ are of small norms.

4.2 General Dependencies

Consider a linear system of equations (1) with general dependencies (2). The coefficients are then nonlinear functions of system parameters. In the proposed approach, the nonlinear functions are estimated by affine forms that enclose their range and a new system with affine linear dependencies is obtained:

$$A(\varepsilon)x(\varepsilon) = b(\varepsilon) \ , \tag{13}$$

where

$$a(\varepsilon)_{ij} = \omega_{ij0} + \omega_{ij}^T\varepsilon, \qquad b(\varepsilon)_i = \omega_{i0} + \omega_i^T\varepsilon \ , \tag{14}$$

and ε ($\varepsilon_i \in [-1, 1]$) is a vector of dummy variables which are considered as new interval parameters. Parametric linear system (13) with affine dependencies (14) can be solved using the Direct Method. The solution of system (13) produced by the Direct method contains the solution of original system (1).

5 Numerical Experiments

Example 1 ([11], [23])

$$\begin{pmatrix} -(p_1+1)p_2 & p_1p_3 & p_2 \\ p_2p_4 & p_2^2 & 1 \\ p_1p_2 & p_3p_5 & \sqrt{p_2} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \ ,$$

$p_1 \in [1, 1.2]$, $p_2 \in [2, 2.2]$, $p_3 \in [0.5, 0.51]$, p_4, $p_5 \in [0.39, 0.4]$.

Table 1 lists the results obtained by the Direct Method, El-Owny's [23] method, Kolev's method [11] and Interval-Affine Gaussian Elimination [1].

Table 1. Comparison of the results for Example 1

Direct Method	El-Owny [23]	Kolev [11]	IAGE [1]
[0.055488, 0.066073]	[0.055479, 0.066083]	[0.055081, 0.066443]	[0.049960, 0.071690]
[0.076315, 0.090294]	[0.076096, 0.090512]	[0.075930, 0.090906]	[0.072889, 0.093943]
[0.557476, 0.606111]	[0.557139, 0.606451]	[0.555988, 0.607462]	[0.556478, 0.607841]

To assess the improvement of the Direct Method over three competing methods, the merit proposed by Kolev ([10]) is used. The 1.55% improvement over El-Owny's method, 6.3% improvement over Kolev's method, and 28.62% over Interval-Affine Gaussian Elimination have been obtained.

Example 2 ([11], [23])

$$\begin{pmatrix} -(p_1+p_2)p_4 & p_2p_4 \\ p_5 & p_3p_5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \ ,$$

p_1, $p_3 \in [0.96, 0.98]$, $p_2 \in [1.92, 1.96]$, p_4, $p_5 \in [0.48, 0.5]$.

Table 2. Comparison of the results for Example 2

Direct Method	El-Owny [23]	Kolev [11]
[0.374762, 0.456527]	[0.374648, 0.456641]	[0.367181, 0.464108]
[1.621595, 1.729273]	[1.621478, 1.729391]	[1.613711, 1.737157]

In this case, the 0.24% improvement over El-Owny's method and 16.2% improvement over Kolev's method have been obtained.

Example 3 ([13], [24]). All matrix coefficients not specified below are equal to zero.

$$\begin{aligned}
&\tfrac{1}{2}a_{11} = a_{12} = a_{21} = a_{65} = -a_{74} = p_1 \ , \\
&a_{22} = 2p_1 + 2p_3,\ a_{33} = 3p_2 + 2p_3,\ a_{66} = p_1 + p_2 \ , \\
&a_{23} = a_{32} = -2p_3,\ a_{68} = p_3,\ a_{86} = p_2 \ , \\
&a_{47} = a_{48} = a_{54} = a_{55} = a_{56} = -a_{61} = -a_{71} = a_{72} = -a_{83} = 1 \ ,
\end{aligned}$$

$$b = \left(0,\ 0,\ -\frac{3}{8}p_4p_2^3,\ 0,\ p_2p_4,\ p_2p_4(p_1 + \frac{1}{2}p_2),\ 0,\ \frac{1}{2}p_2^2p_4\right)^T \ .$$

First, the system is solved for 1% uncertainty in all parameters, that is

$$p = ([0.0995,\ 1.005],\ [0.0995,\ 1.005],\ [0.74625,\ 0.75375],\ [9.95,\ 10.05])^T \ .$$

Interval estimates of the solution set are summarised in Tab. 3. The table includes the estimates produced by the Direct Method, the results of Popova's parametric fixed-point iteration [24], and the hull solution. The results of the Direct Method and the fixed-point iteration show good sharpness of the enclosure for all solution components. The quality of both estimates is comparable.

Table 3. Comparison of the results for Example 3 with 1% uncertain parameters

Direct Method	Popova [24]	Hull
[0.24464, 0.25537]	[0.24470, 0.25537]	[0.24479, 0.25529]
[−0.51070, −0.48934]	[−0.51070, −0.48945]	[−0.51059, −0.48958]
[−1.01726, −0.98281]	[−1.0173, −0.98304]	[−1.01710, −0.98309]
[−0.76990, −0.73016]	[−0.76990, −0.73032]	[−0.76973, −0.73072]
[6.66883, 6.83118]	[6.6691, 6.8312]	[6.66989, 6.83089]
[3.95951, 4.04054]	[3.9599, 4.0406]	[3.96010, 4.04010]
[−0.6860, −0.64738]	[−0.6860, −0.64887]	[−0.68420, −0.64953]
[0.64738, 0.6860]	[0.64887, 0.6860]	[0.64953, 0.68420]

Next table lists the solutions for 2% uncertainties in three first parameters and 30% uncertainty in last parameter.

The results of the Popova's method are slightly narrower which requires some comment. This difference is mainly due to the method of bounding nonlinear

Table 4. Comparison of the results for Example 3 with 2% uncertain lengths and 30% uncertain load

Direct Method	Popova [24]	Hull
[0.201774, 0.298301]	[0.20409, 0.29831]	[0.205777, 0.296805]
[−0.595828, −0.404322]	[−0.59583, −0.40897]	[−0.593610, −0.411554]
[−1.181037, −0.819263]	[−1.1811, −0.82856]	[−1.177780, −0.829733]
[−0.902773, −0.597452]	[−0.90278, −0.60442]	[−0.899410, −0.611219]
[5.623301, 7.876724]	[5.6390, 7.8768]	[5.658540, 7.870740]
[3.340685, 4.659515]	[3.3618, 4.6596]	[3.366000, 4.646000]
[−0.850811, −0.482722]	[−0.85082, −0.49464]	[−0.799475, −0.543306]
[0.482722, 0.850811]	[0.49464, 0.85082]	[0.543306, 0.799475]

functions. Popova uses generalized interval arithmetic, while Direct Method uses affine arithmetic. Generalized interval arithmetic usually gives narrower enclosures. For example, consider the following expression:

$$z(p) = -0.12 + 0.0192(p_1 + p_2)/p_4 + 0.0048p_2p_4 - 0.0224p_3/p_5 + 0.0896p_5 \ ,$$

with p_1, $p_3 \in [0.96, 1.04]$, $p_2 \in [1.92, 2.08]$, p_4, $p_5 \in [0.48, 0.52]$. Generalized interval arithmetic gives $z(\boldsymbol{p}) = [-0.0143946, 0.0148305]$, while affine arithmetic gives $z(\boldsymbol{p}) = [-0.0149484, 0.0150612]$. However, computing bounds with generalized interval arithmetic requires some additional calculations, the global monotonicity of the function with respect to all variables has to be checked. Affine calculations does not require any additional effort and hence performs faster, and thus the overall method performs faster, approximately five times faster in the first case and three times faster in the second case. It is worth to mention that in case of parametric linear systems with affine-linear dependencies, the Direct Method gives narrower results then the fixed-point iteration.

6 Conclusions

The goal of this work was to propose a highly automated efficient method for solving parametric linear systems with nonlinear dependencies and to study the behaviour of the presented approach. Using several examples, it was demonstrated that the proposed Direct Method is superior to other considered (known in the literature) methods based on affine or generalized affine arithmetic.

The Direct Method have been also compared with the interval fixed-point iteration combined with generalized interval arithmetic for range enclosure. The results of the fixed-point iteration are slightly narrower which is mainly due to the range enclosure technique. In case of affine-linear dependencies, the Direct Method produces narrower enclosures than the fixed-point iteration. Nevertheless, the results of the Direct Method based on affine arithmetic are comparable with the results of the fixed-point iteration, while the presented method performs faster. Moreover, the Direct Method combined with affine arithmetic can

be easily implemented using available software and can handle different types of nonlinear dependencies. Combining this method with more sophisticated tool for range enclosure would probably result in a very powerful method for solving parametric linear systems with complicated dependencies.

The method yield validated inclusions computed by a finite precision arithmetic. The present approach, is also applicable to other uncertainty theories such as fuzzy set theory or random set theory.

References

1. Akhmerov, R.R.: Interval-affine Gaussian algorithm for constrained systems. Reliable Computing 11(5), 323–341 (2005)
2. Alefeld, G., Frommer, A., Lang, B. (eds.) Scientific Computing and Validated Numerics. Mathematical Research 90. Proceedings of the International Symposium on Scientific Computing Computer Arithmetic and Validated Numerics (SCAN-95), Wuppertal, Germany. Akademie-Verlag (1996)
3. Chen, S., Lian, H., Yang, X.: Interval static displacement analysis for structures with interval parameters. International Journal for Numerical Methods in Engineering 53(2), 393–407 (2001)
4. Fang, C.F., Chen, T., Rutenbar, R.: Floating-point error analysis based on affine arithmetic. In: Proc. 2003 International Conf. on Acoustic, Speech and Signal Processing, Part II, vol. 2, pp. 561–564 (2003)
5. Hansen, E.R.: A Generalised Interval Arithmetic. LNCS, vol. 29, pp. 7–18. Springer, Heidelberg (1975)
6. Kearfott, R.B.: Decomposition of arithmetic expressions to improve the behaviour of interval iteration for nonlinear systems. Computing 47(2), 169–191 (1991)
7. Kolev, L.V.: An Improved Interval Linearization for Solving Nonlinear Problems. Numerical Algorithms 37(1-4), 213–224 (2004)
8. Kolev, L.V.: New Formulae for Multiplication of Intervals. Reliable Computing 12(4), 281–292 (2006)
9. Kolev, L.V.: Automatic Computation of a Linear Interval Enclosure. Reliable Computing 7(1), 17–28 (2001)
10. Kolev, L.V.: A method for outer interval solution of linear parametric systems. Reliable Computing 10(3), 227–239 (2004)
11. Kolev, L.V.: Solving Linear Systems whose Elements are Non-linear Functions of Intervals. Numerical Algorithms 37, 199–212 (2004)
12. Kramer, W.: Generalized Intervals and the Dependency Problem. In: PAMM 2007, vol. 6(1), pp. 683–684 (2007)
13. Kulpa, Z., Pownuk, A., Skalna, I.: Analysis of linear mechanical structures with uncertainties by means of interval methods. Computer Assisted Mechanics and Engineering Sciences 5(4), 443–477 (1998)
14. Kurzhanski, A.B.: Ellipsoidal calculus for uncertain dynamics. In: Abstracts of the International Conference on Interval and Computer-Algebraic Methods in Science and Engineering (INTERVAL/1994) p. 162, St. Petersburg, Russia (1994)
15. Messine, F.: Extentions of Affine Arithmetic: Application to Unconstrained Global Optimization. Journal of Universal Computer Science 8(11), 992–1015 (2002)
16. Miyajima, S., Miyata, T., Kashiwagi, M.: On the Best Multiplication of the Affine Arithmetic. Transactions of the Institute of Electronics, Information and Communication Engineers J86-A(2), 150–159 (2003)

17. Miyajima, S., Kashiwagi, M.: A dividing method utilizing the best multiplication in affine arithmetic. IEICE Electronics Express 1(7), 176–181 (2004)
18. Moore, R., Hansen, E., Leclerc, A.: Rigorous methods for global optimization. In: Recent Advances in Global Optimization, pp. 321–342. Princeton University Press, Princeton (1992)
19. Muhanna, R.L., Erdolen, A.: Geometric uncertainty in truss systems: an interval approach. In: Muhannah, R.L. (ed.) Proceedings of the NSF Workshop on Reliable Engineering Computing (REC): Modeling Errors and Uncertainty in Engineering Computations, Savannah, Georgia USA, pp. 239–247 (2006)
20. Nedialkov, N.S., Kreinovich, V., Starks, S.A.: Interval Arithmetic, Affine Arithmetic, Taylor Series Methods: Why, What Next? Numerical Algorithms 37(1-4), 325–336 (2004)
21. Neumaier, A.: Interval Methods for Systems of Equations. In: Encyclopedia of Mathematics and its Applications, pp. xvi + 255. Cambridge University Press, Cambridge (1990)
22. Neumaier, A., Pownuk, A.: Linear Systems with Large Uncertainties, with Applications to Truss Structures. Reliable Computing 13(2), 149–172 (2007)
23. El-Owny, H.: Parametric Linear System of Equations, whose Elements are Nonlinear Functions. In: 12th GAMM - IMACS International Symposion on Scientific Computing, Computer Arithmetic and Validated Numerics, pp. 26–29 (2006)
24. Popova, E.D.: On the Solution of Parametrised Linear Systems. In: Kraemer, W., von Gudenberg, J.W. (eds.) Scientific Computing, Validated Numerics, Interval Methods, pp. 127–138. Kluwer Acad. Publishers, Dordrecht (2001)
25. Popova, E.D.: Solving Linear Systems Whose Input Data Are Rational Functions of Interval Parameters. In: Boyanov, T., Dimova, S., Georgiev, K., Nikolov, G. (eds.) NMA 2006. LNCS, vol. 4310, pp. 345–352. Springer, Heidelberg (2007)
26. Rohn, J.: A Method for Handling Dependent Data in Interval Linear Systems. Academy of Science of the Czech Republic, Institute of Computer Science, Technical report No. 911 (2004)
27. Rump, S.M.: Verification Methods for Dense and Sparse Systems of Equations. In: Herzberger, J. (ed.) Topics in Validated Computations – Studies in Computational Mathematics, pp. 63–136. Elsevier, Amsterdam (1994)
28. Shary, S.P.: Solving tied interval linear systems. Siberian Journal of Computational Mathematics 7(4), 363–376 (2004)
29. de Figueiredo, L.H., Stolfi, J.: Self-Validated Numerical Methods and Applications. Brazilian Mathematics Colloquium monograph (1997)
30. Skalna, I.: A method for outer interval solution of systems of linear equations depending linearly on interval parameters. Reliable Computing 12(2), 107–120 (2006)
31. Skalna, I.: Evolutionary optimization method for approximating the solution set hull of parametric linear systems. In: Boyanov, T., Dimova, S., Georgiev, K., Nikolov, G. (eds.) NMA 2006. LNCS, vol. 4310, pp. 361–368. Springer, Heidelberg (2007)
32. Vu, X., Sam-Haroud, D., Faltings, B.V.: A Generic Scheme for Combining Multiple Inclusion Representations in Numerical Constraint Propagation. Artificial Intelligence Laboratory, Swiss Federal Institute of Technology in Lausanne (EPFL), Technical Report No. IC/2004/39 (2004)
33. http://www.ic.unicamp.br/ stolfi/EXPORT/projects/affine-arith/Welcome.html

Evaluating Lava Flow Hazard at Mount Etna (Italy) by a Cellular Automata Based Methodology

Maria Vittoria Avolio[1], Donato D'Ambrosio[1], Valeria Lupiano[2], Rocco Rongo[2], and William Spataro[1,3,⋆]

[1] Department of Mathematics and High Performance Computing Center, University of Calabria, Italy
[2] Department of Earth Sciences and High Performance Computing Center, University of Calabria, Italy
[3] Department of Mathematics, University of Calabria, Via Pietro Bucci, I-87036 Rende, Italy
spataro@unical.it

Abstract. The individuation of areas that are more likely to be interested by lava eruptions is of fundamental relevance for mitigating possible consequences, both in terms of loss of human lives and material properties. Here we show a methodology for defining flexible high-detailed lava invasion susceptibility maps. It relies on both an adequate knowledge of the volcano, assessed by an accurate analysis of its past behaviour, a reliable Cellular Automata model for simulating lava flows on present topographic data and on High Performance Parallel Computing for increasing computational efficiency. The application of the methodology to the case of Mt Etna, the most active volcano in Europe, points out its usefulness in land use planning and Civil Defence applications.

Keywords: Cellular Automata, Simulation, Lava flows, Hazard Assessment.

1 Introduction

Many volcanic areas around the world are densely populated and urbanized. In Italy, Mt Etna is home to approximately one million people, though being the most active volcano in Europe [1]. More than half of the events occurred in the last four centuries report damage to human properties in numerous towns on the volcano flanks. In particular, eruptions in 1669 and 1928 destroyed entire villages and, in the earlier case, portions of Catania, the main city of the region [2]. In last decades, the vulnerability of the Etnean area has increased exponentially due to continued, sometimes wild, urbanization, with the consequence that new eruptions may involve even greater risks. In recent crises, countermeasures based on embankments or channels were adopted to stop or deflect lava [3]. In some

⋆ Corresponding Author.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 495–504, 2010.

cases, even exceptional actions were necessary which, for instance, involved the adoption of explosive and/or cement tapping inside the active craters in order to reduce lava emission. Such kind of interventions are generally performed while the eruption is in progress, by both not guarantying their effectiveness, and by inevitably putting into danger the safety of involved persons. Lava flows forecasting could significantly change this scenario. A modern and widely adopted approach is the application of algorithms that permit numerical simulations of lava flows ([4] and [5]), for the purpose of individuating affected areas in advance. For instance, in 2001 the path of the eruption that threatened the town of Nicolosi on Mt Etna was correctly predicted by means of a lava flows simulation model [6], providing useful information for Civil Defence. However, this *modus operandi* does not take into account the fact that an eruption can have different characteristics (e.g. different flow-rates). As a matter of fact, an *a priori* knowledge of the degree of exposure of the volcano surrounding areas is desirable in order to allow both the realization of preventive countermeasures, and a more rational land use planning. Here we show results of a new methodology for the definition of flexible high-resolution lava invasion susceptibility maps, and show results and applications related to the South-Eastern flank of Mt Etna, the most densely populated sector of the volcano.

2 An Integrated Methodology for Lava Flows Impact Prediction

The methodology here proposed relies on the application of a lava flows computational model for simulating new events on present topographic data, and on a new criterion for evaluating the impact of performed simulations in terms of spatial hazard. The computational model should be well calibrated and validated and thus able to reproduce the events which characterise the considered volcano with a high level of accuracy. Furthermore, it should be as much efficient as possible since, depending on the extent of the considered area, a great number of simulation could be required.

An accurate analysis of the past behaviour of the volcano is required for the purpose of classifying the events that historically interested the region. By assuming that the volcano's behaviour will not dramatically change in the near future, a representative case is selected for each of the individuated classes and triggered from each crater of a grid which opportunely covers the area. In such a way, a meaningful database of plausible simulated lava flows can be obtained, by characterising the study area both in terms of areal coverage, and lava flows typologies.

Subsequently, such data are processed by considering a proper criterion of evaluation. It is worth to note that a first solution could simply consist in considering lava flows overlapping, by assigning a greater hazard to those sites interested by a higher number of simulations. However, a similar choice could be misleading. In fact, depending on their particular traits (such as the location of the main crater, the duration and the amount of emitted lava, or the effusion

rate trend), different events can occur with different probabilities, which should be taken into account in evaluating the actual contribution of performed simulations with respect to the definition of the overall susceptibility of the study area. In most cases, such probabilities can be properly inferred from the statistical analysis of past eruptions, allowing for the definition of a more refined evaluation criterion. Accordingly, in spite of a simple hitting frequency, a measure of lava invasion susceptibility can be obtained in probabilistic terms. In the following, we show how such approach was applied to the South-Eastern flank of Mt Etna.

3 Lava Invasion Map Compilation

Natural complex phenomena such as lava flows are usually difficult to simulate using traditional methods based on differential equations systems [7]. Among alternative methods, usually based on heuristic and approximation techniques, Cellular Automata (CA) [8] are gaining the attention of the International Scientific Community for their robustness, reliability and range of applicability, despite their apparent simplicity [9]. Cellular Automata applications are very broad, ranging from fluid-dynamics [10] to traffic control [11], from theoretical issues [8] to image processing [12], from cryptography [13] to economics and social sciences [14]. In particular, applications concerning fluid-dynamics of geological processes have produced interesting and considerable results in the field of simulation of lava, landslide and pyroclastic flows. In the classic view of CA systems, space and time are discrete. They are based on a regular division of the space in regular cells. Time is sub-divided in steps, which may be considered as constant time intervals. Each cell embeds an identical computational element whose state specifies the cell condition throughout a computational step. The overall dynamics of the system emerges as the result of the simultaneous application, at discrete time steps, of proper "local rules" of evolution to each cell of the CA.

Our research group is experienced in this field since 1982, when a first computational model of basaltic lava flows was proposed [15]. In recent years we enriched the SCIARA family of lava flows simulation models, by proposing improved releases and applying them to the simulation of diverse Etnean cases of study [6]. For the purposes of this work, being the South-Eastern flank of mount Etna a vast area of about $237km^2$ (cf. Fig. 1), a great number of simulation were originally planned to be executed (cf. below) and thus a computational model that is at the same time accurate and efficient seemed to be the best choice. For this reason, the SCIARA-fv release (simply SCIARA in the following), based on the Cellular Automata computational paradigm and, specifically, on the Macroscopic Cellular Automata approach for the modelling of spatially extended dynamical systems [16], was chosen as the reference computational algorithm.

In order to execute a simulation, SCIARA requires a hexagonal discretisation of the topography over which the event will be simulated (for the purpose

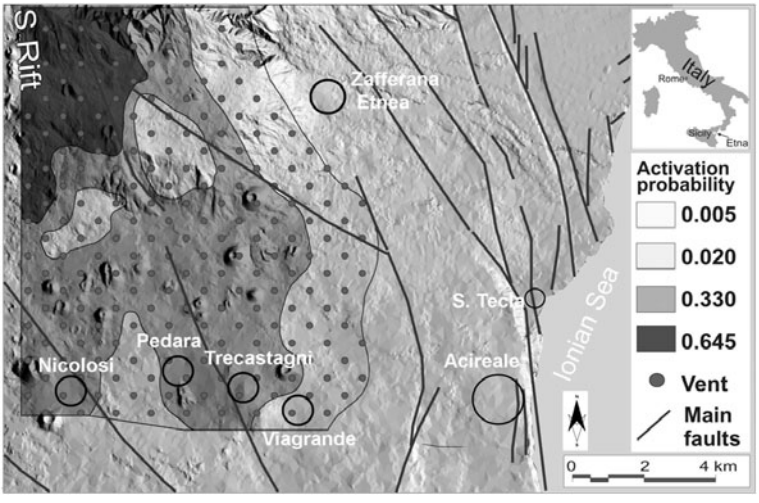


Fig. 1. The South-Eastern flank of Mt Etna. Grey colouring indicate crater activation probability, in decreasing order, corresponding to 0.645, 0.33, 0.02, 0.005 and 0, respectively; dots represent the grid of 208 craters considered in this work.

of minimising the intrinsic anisotropic effects of discrete modelling of continuous systems), the craters coordinates and the emission rate history, besides the specification of a set of parameters related to the magma rheology.

Accordingly, the considered Etnean sector was digitalised as a DEM (Digital Elevation Model) of 2272×1790 hexagonal cells, each with a $5m$ apothem, by considering the GIS vectorialization of good-quality $1 : 10000$ scale topographic maps. A proper calibration and validation phases, carried out by the adoption of a Parallel Master Slave Genetic Algorithm, helped in devising a suitable set of model's parameters, by allowing the model to reproduce Etnean lava flows with a high level of accuracy [17]. At the same time, many computational refinements were implemented, significantly speeding up the model. Still, even a sensitivity analysis study was performed, by excluding unpredictable changes in simulation outcomes when small changes are considered in input data, thus demonstrating the overall robustness of the computational algorithm.

Eventually, with the exception of few isolated cases, a typical effusive behaviour was strongly evidenced by the analysis of the volcano past activity. As a consequence, we suppose that such behaviour will not dramatically change in the near future and thus that the SCIARA lava flows simulation model, calibrated and validated on a set of effusive eruptions, is adequate for simulating new events on Mt Etna.

3.1 Study of the Volcano

As specified above, the proposed methodology involves a preliminary study of the past activity of the volcano for the purpose of characterising the study area in

terms of eruptive behaviour. Accordingly, such study was performed by analysing a significant set of 64 events occurred on Mt Etna since 1600 AD, from which information is quite reliable [1], classified on the basis of both their duration and the amount of emitted lava, and giving rise to 50 not empty classes. For each of them, values were normalized into the range [0,1], obtaining the probability, p_c, that a new hypothetical event can occur, depending on which class it belongs to. Eventually, 50 representative cases were selected, one for each of the individuated classes. As regards the effusion rate trend to be considered, and on the basis of performed analysis of past eruptions, two typical effusion rate trends for Etnean lava flows were taken into account. In particular, both typologies can be considered equi-probable and well approximated by Gaussian-like distributions with maximum flow rate values at 1/6 and 1/3 of the total duration. Accordingly, a trend probability, p_t, was defined and imposed to the constant value of 0.5 for both the individuated trends. Consequently, by considering the combination of the 50 representative events in terms of emitted-lava/duration and the two representative effusion rate trends, a total of 100 representative typologies of lava flows were obtained which we assumed adequate for representing all the possible behaviour of the volcano from the emissive point of view. Further details regarding the classification procedure and effusion rate determination can be found in [18].

Further, depending on their position, different sites can have different probability to trigger new lava flows. Many volcanological and geo-structural (sometimes conflicting) studies attempted to individuate Etnean sectors that are more prone to generate new eruptive events. Among these, Behncke et al. [1], proposed a characterisation of the study area in terms of probability of activation of new craters by statistically analysing historical eruptions in terms of spatial distribution of eruptive fractures, vent density and their concentration in rift zones, tectonic structures and proximity to the summit area of the volcano (where eruptions are statistically more frequent). The result of such study is shown in Fig. 1, where the Southern-East flank of Mt Etna is subdivided into five sectors, representing areas characterised by different probabilities of activation of new craters. For the purpose of this study, in agreement with the authors of the original research, such probabilities were assumed to be 0.645, 0.33, 0.02, 0.005 and 0, respectively, by considering the vent and fracture density of each area. Accordingly, a probability of activation for a generic crater in the study area, p_s, was defined, which assumes one of the above specified values depending on which sector it is located in (cf. Fig. 1). In short, we made the fundamental assumption that events on Mt Etna can be classified on the basis of the following peculiar features: the amount of emitted lava and the event duration, the emission rate trend, as well as the source position. Moreover, for each of the above cited features, characteristic probabilities were defined on the basis of the statistical analysis of past eruptions. Consequently, if a new event is conjectured to be triggered in the study area, an overall (conditioned) probability of occurrence, p_e, can be defined by simply

considering the product of the individual probabilities of its characteristic features:

$$p_e = p_c \cdot p_t \cdot p_s \tag{1}$$

3.2 Lava FLOW Hazard Susceptibility at Mt Etna

Once representative lava flows were devised in terms of both volume/duration and effusion rate trend, a set of simulations were planned to be executed in the study area by means of the SCIARA lava flows simulation model. At this purpose, a grid composed of 208 craters was defined on the considered SE flank of Mt Etna, as shown in Fig. 1. It is composed of two $1km$ spaced sub grids, the latter displaced by $500m$ along South and East directions with respect to the former. This choice allowed to both adequately and uniformly cover the study area, besides considering a relatively small number of craters.

It is worth to note that, as well as representative lava flows can be characterised by the conditioned probability $p_c \cdot p_t$ (being p_c the probability related to the event's membership class, and p_t the probability related to the event's emission rate trend), still a crater in the grid can be characterised by the probability of activation p_s, depending on in which sector it belongs to (cf. Fig. 1). In this way, a representative lava flow on Mt Etna can be simulated by considering a given point of the defined grid as source location, by evaluating its probability of occurrence by simply applying equation 1.

Therefore, all the 100 representative lava flows were simulated for each of the 208 craters of the grid, thus obtaining a set of $N = 20800$ simulations. By considering the extent of the study area and the duration of the events to be simulated (which ranges from 15 to 500 days), computing times would have resulted in the order of about one year on standard sequential architectures, in spite of the high computational efficiency of the employed lava flow simulation model. Accordingly, the simulation phase was performed on two high performance parallel machines, namely a 16 Itanium processor NEC TX7 NUMA super computer and a 4 bi-quadcore G5 processor Apple Xserve cluster, thus reducing the overall execution to less than one month. [18] shows preliminary results of the proposed methodology, which is described in the following paragraphs.

Lava flow invasion susceptibility was then punctually evaluated by considering the contributions of all the simulations which affected a generic site in terms of their probability of occurrence, with a resulting detail which exclusively depends on the resolution of the considered DEM. Thus, if a given DEM cell of coordinates x, y was affected by $n_{x,y} \leq N$ simulations, its susceptibility was defined as the sum of the probabilities of occurrence of involved lava flows, $p_e^{(i)} (i = 1, 2, \ldots, n_{x,y})$:

$$s_{x,y} = \sum_{i=1}^{n_{x,y}} p_e^{(i)} \tag{2}$$

Thus, on the basis of the previous assumptions, the following pseudo-code can be adopted for determining lava invasion susceptibility for a certain area:

```
lava_invasion_computation(){
  for each DEM cell (x,y)
    compute number simulations n(x,y) affecting cell (x,y)
    for i=0 to n(x,y)
          s(x,y) = s(x,y) + pe(i)
}
```

where: pe(i) is the probability of occurrence (as defined in section 3.1) of flow i and s(x, y) the lava invasion susceptibility of cell with coordinates (x, y). The resulting final lava invasion susceptibility map, shown in Fig. 2, represents the probability that future events will affect the study area. A similar map results particularly suitable for land use planning purposes, since it gives an overall information about the vulnerability of the entire study area with respect to the occurrence of new lava flows, independently from their effective source locations and emissive behaviours.

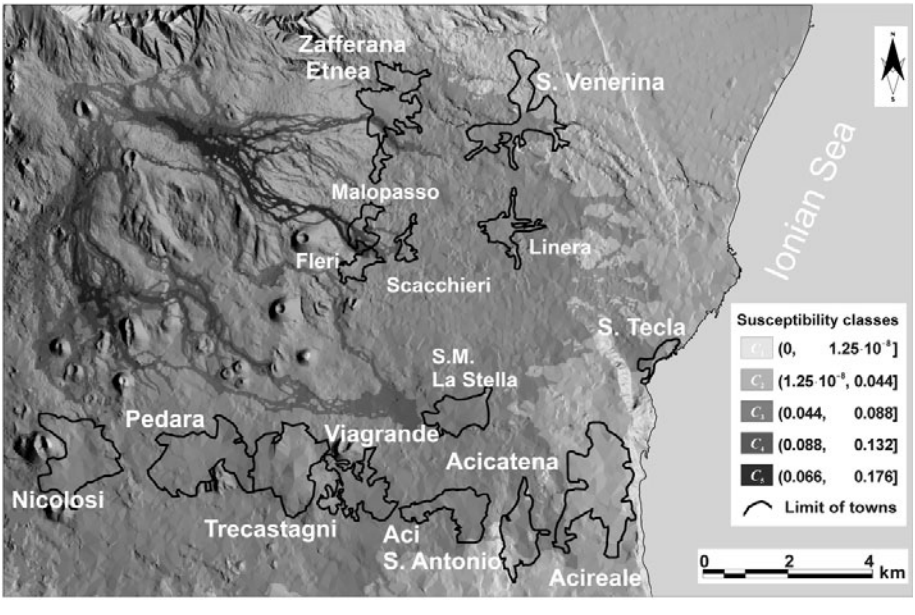


Fig. 2. Hazard map of the study area based on the 20,800 simulations. As a compromise between map readability and reliability, 5 classes are reported (grey colouring), in increasing order of susceptibility (probability of lava invasion).

Eventually, since the obtained results are strongly related to morphological conditions, they could require to be updated each time topographic alterations occur. In this case, it will be sufficient to consider a DEM representing the actual

topography, and re-simulate only the (generally few) representative events which involve the modified areas. A new susceptibility map can then be obtained by simply reprocessing the new set of simulations, which is a quite rapid procedure even on sequential computers. At the contrary, if a certain number of events will occur on Mt. Etna, whose characteristics determine a modification of the previously individuated representative set of lava flows, a new overall simulation phase will be required in order to obtain a correct susceptibility scenario.

3.3 Civil Defence Applications

The availability of a large number of scenarios of different eruption types, magnitudes and locations simulated for this study allows an instantaneous extraction of various scenarios on demand, since they are already stored in the simulation data base. This would be especially relevant once premonitory signs such as localized seismicity and ground deformation indicate the possible site of an imminent eruption.

Such scenarios may concern eruptions from single vents, which would in the ideal case coincide with the hypothetical vents of the simulation grid. If the

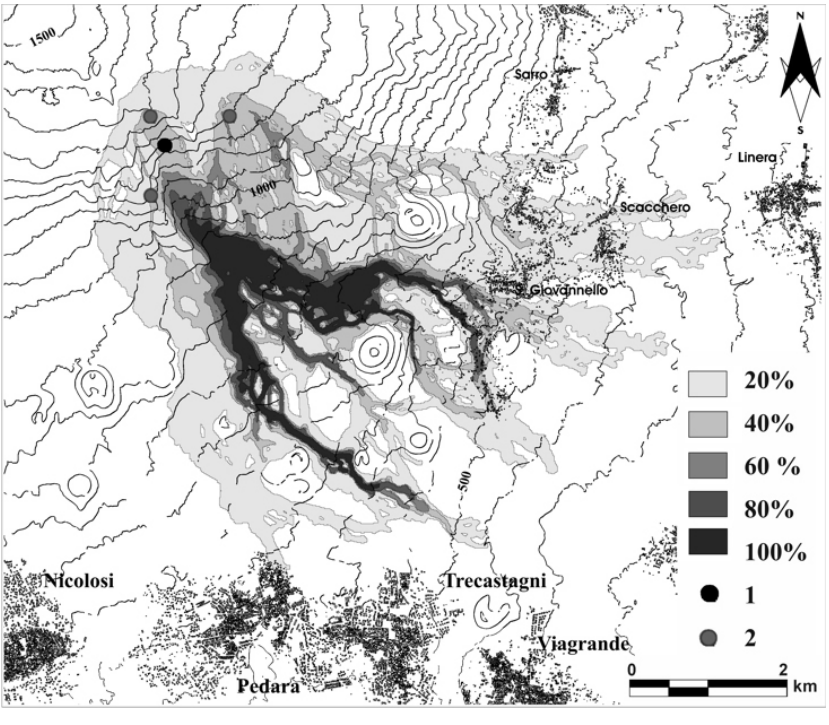


Fig. 3. Hazard map related to a single hypothetical event of unknown emission rate. In grey the probability (in percentage terms) of lava invasion. Key: 1) real lava source crater 2) the 3 nearest craters to the lava source point.

actual vent does not coincide with any of those vents, it would be possible to extract the combined lava flow scenarios from the nearest three hypothetical vents next to the true one (Fig 3).

Further Civil Defence oriented applications are possible. For instance, our methodology permits to identify all source areas of lava flows capable of affecting a given area of interest, such as a town or a significant infrastructure. This application is rapidly accomplished by a reprocessing of the simulation set, by simply eliminating the events that do not affect the area of interest and by circumscribing the source locations of the remaining events. This kind of application allows to rapidly assess the threat posed by an eruption from a given area and thus represents a useful tool for decision support.

4 Conclusions

The fundamental problem of assessing the impact of future lava eruptions on a vast volcanic region is a crucial aspect for risk mitigation. This paper shows the application of a new kind of criterion for the compilation of lava invasion susceptibly maps, based on Cellular Automata and Parallel Computing, applied to the South-Eastern flank of Mt Etna (South Italy).

Different Civil Defence applications can be devised on the proposed methodology. For instance, it is possible to identify all source areas of lava flows capable of affecting a given area of interest. Still, a specific category of simulation can be dedicated to the assessment of protective measures, such as earth barriers, for mitigating lava invasion susceptibility in given areas.

Even if a more rigorous assessment of the reliability of the methodology is certainly desirable for effective usage in Civil Defense - such as the compilation of the map on a subset of sample events (e.g. occurred in the first 300 years) and validating it over the remaining ones - the proposed method seems to be more reliable when compared with a more classical criterion of hazard mapping. Current work regards the application of the methodology to other areas of the volcano and a rigorous study on a map validation procedure.

Acknowledgments. Authors thank Dr. B. Behncke and Dr. M. Neri from the Istituto Nazionale di Geofisica e Vulcanologia of Catania (Sicily, Italy), who provided topographic maps and the volcanological data. The authors are also grateful to Prof. G.M. Crisci for his precious comments and the common researches.

References

1. Behncke, B., Neri, M., Nagay, A.: Lava flow hazard at Mount Etna (Italy): New data from a GIS-based study. Spec. Pap. Geol. Soc. Am. 396, 187–205 (2005)
2. Chester, D.K., Duncan, A.M., Dibben, C., Guest, J.E., Lister, P.H.: Mascali, Mount Etna Region Sicily: An Example of Fascist Planning During the 1928 Eruption and Its Continuing Legacy. Nat. Hazards 19, 29–46 (1999)

3. Barberi, F., Brondi, F., Carapezza, M.L., Cavarra, L., Murgia, C.: Earthen barriers to control lava flows in the 2001 eruption of Mt. Etna. J. Volcanol. Geotherm. Res. 123, 231–243 (2003)
4. Ishihara, K., Iguchi, M., Kamo, K.: Lava flows and domes: emplacement mechanisms and hazard implications. In: IAVCEI Proceedings, pp. 174–207. Springer, Heidelberg (1990)
5. Del Negro, C., Fortuna, L., Herault, A., Vicari, A.: Simulations of the 2004 lava flow at Etna volcano using the magflow cellular automata model. Bull. Volcanol. 70, 805–812 (2008)
6. Crisci, G., Rongo, R., Di Gregorio, S., Spataro, W.: The simulation model SCIARA: the 1991 and 2001 lava flows at Mount Etna. J. Volcanol. Geotherm. Res. 132, 253–267 (2004)
7. McBirney, A.R., Murase, T.: Rheological properties of 731 magmas. Annu. Rev. Earth Planet. Sci. 12, 337–357 (1984)
8. Von Neumann, J.: Theory of self reproducing automata. Univ. Illinois Press, Urbana (1966)
9. Chopard, B., Droz, M.: Cellular Automata Modeling of Physical Systems. Cambridge University Press, Cambridge (1998)
10. Succi, S.: The Lattice Boltzmann Equation for Fluid Dynamics and Beyond. Oxford University Press, Oxford (2004)
11. Di Gregorio, S., Festa, D., Rongo, R., Spataro, W., Spezzano, G., Talia, D.: A microscopic freeway traffic simulator on a highly parallel system. In: D'Hollander, F.P.E.H., Joubert, G.R., Trystam, D. (eds.) Parallel Computing: State-of-the-Art and Perspectives, pp. 69–76 (1996)
12. Rosin, P.L.: Training Cellular Automata for Image Processing. IEEE Trans. on Image Process. 15(7), 2076–2087 (2006)
13. Tomassini, M., Perrenoud, M.: Cryptography with cellular automata. Appl. Soft Comput. 1(2), 151–160 (2001)
14. Page, S.: On Incentives and Updating in Agent Based Models. Comput. Econ. 10(1), 67–87 (1997)
15. Crisci, G.M., Di Gregorio, S., Ranieri, G.: A cellular space model of basaltic lava flow. In: Proceedings Int. Conf. Applied Modelling and Simulation 82, Paris-France, vol. 11, pp. 65–67 (1982)
16. Di Gregorio, S., Serra, R.: An empirical method for modelling and simulating some complex macroscopic phenomena by cellular automata. Fut. Gener. Comp. Syst. 16, 259–271 (1999)
17. Rongo, R., Spataro, W., D'Ambrosio, D., Avolio, M.V., Trunfio, G.A., Di Gregorio, S.: Lava flow hazard evaluation through cellular automata and genetic algorithms: an application to Mt. Etna. volcano. Fund. Inform. 8, 247–268 (2008)
18. Crisci, G.M., Iovine, G., Di Gregorio, S., Lupiano, V.: Lava-flow hazard on the SE flank of Mt. Etna. (Southern Italy). J. Volcanol. Geotherm. Res. 177(4), 778–796 (2008)

Application of CoSMoS Parallel Design Patterns to a Pedestrian Simulation

Sarah Clayton, Neil Urquhard, and Jon Kerridge

School of Computing, Edinburgh Napier University, Merchiston Campus, Edinburgh EH10 5DT, United Kingdom

Abstract. In this paper, we discuss the implementation of a simple pedestrian simulation that uses a multi agent based design pattern developed by the CoSMoS research group. Given the nature of Multi Agent Systems (MAS), parallel processing techniques are inevitably used in their implementation. Most of these approaches rely on conventional parallel programming techniques, such as threads, Message Passing Interface (MPI) and Remote Method Invocation (RMI). The CoSMoS design patterns are founded on the use of Communicating Sequential Processes (CSP), a parallel computing paradigm that emphasises a process oriented rather than object oriented programming perspective.

1 Introduction

The realistic simulation of pedestrian movement is a challenging problem, a large number of individual pedestrians may be included in the simulation. Rapid decisions must be made about the trajectory of each pedestrian in relation to the environment and other pedestrians in the vicinity. Parallel processing has allowed such simulations to include tens of thousands of pedestrian processes travelling across large areas. The multi-agent systems paradigm has recently been used to significant effect within pedestrian simulation, typically each pedestrian within the simulation equates to a specific agent. Examples of these are described in [1] [2]. In [2], the parallel aspect of the simulation is implemented using MPI.

The Complex Systems Modelling and Simulation infrastructure (CoSMoS) research project [3], a successor to the TUNA [4] [5] research project, is an attempt to develop reusable engineering techniques that can be applied across a whole range of MAS and simulations. Examples include three dimensional simulations of blood clot formation in blood vessels, involving many thousands of processes running across a number of networked computers [6].

The aims of the CoSMoS environment are to provide a 'massively-concurrent and distributed' [7] system through the use of the process-oriented programming model. The process-oriented paradigm derives from the concept of CSP [8] and pi-calculus. The purpose of these methods is the elimination of perennial problems in programming using conventional parallel techniques, such as threads. These problems include deadlock, livelock and race hazards.

Examples produced by the CoSMoS research group are implemented using the language occam-pi, a language developed to write programmes that comply

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 505–512, 2010.

with the principles of both CSP and pi-calculus. To this end, the Kent Retargetable occam Compiler (KRoC) [9] was developed at the University of Kent Computing Laboratory. Similar language features are also provided by the more accessible Java Communicating Sequential Processes (JCSP) application programming interface (API), also developed at the University of Kent [10]. This provides a means of implementing the semantics of CSP, that uses the underlying Java threading model, without the developer needing to be concerned with this. Although most of the examples from CoSMoS are developed using KRoC, the work described here has been created using JCSP.

2 CSP and pi-Calculus

In this section, we give a brief overview of CSP and pi-calculus, and part of the rich set of semantics for concurrent programming that they offer. These are available as language features in the occam-pi programming language, and the JCSP API. The work done by the CoSMoS research group [3] has mostly been done using occam-pi. However, the work done here is based on JCSP, although the language primitives are more or less equivalent between the two [10].

2.1 Processes

The main difference between CSP and traditional programming approaches is that it is process-oriented rather than object-oriented. Instead of separate classes, computational tasks are divided into processes. Unlike objects, these processes entirely encapsulate their data and maintain their own state. These cannot be altered by other processes [11].

2.2 Channels and Barriers

Instead of being propagated by listeners, events in CSP based environments are communicated using channels and barriers. Barriers are a fundamental part of Bulk Synchronous Processing (BSP) [12]. They are analogous to events in in CSP [8]. They allow multiple and heterogeneous processes to synchronise their activities, and enforce lockstep parallelism between them. When a process synchronises on a Barrier, it waits until all other processes enrolled on the Barrier have also synchronised before continuing processing.

Processes communicate between each other using channels. Channels are synchronous. Any process writing to a channel will block until the reader is able to process the message. Although it is possible to implement buffering in these channels using JCSP, no buffered channels are used here. One2OneChannels allow point to point communication between processes, and Any2OneChannels allow many processes to communicate with a single process. Other channel types are available but are not discussed here.

3 Implementation

The simulation is at a fine grained level of granularity, and is highly scalable. It has three main elements, agents, sites, and an overall site server. These are implemented as processes rather than passive objects, and communicate with each other using channels, rather than through method calls. This allows processes and their data to be completely encapsulated. Any change of state or control information is communicated to other process through channels.

The space to be simulated is modelled by multiple Site processes, in a way similar to the one described in [5]. A Site process acts as a server to any number of Agent client processes. The use of a client-server architecture here eliminates the dangers of deadlock.

Agents are mobile across sites, they may deregister themselves from any given Site and migrate to another. Each Site has a register channel for this purpose, where it receives registration requests from Agents, in the form of their server channel ends. This allows communication to be immediately established between the Agent as client and the Site as server. The SiteServer operates in a manner conceptually similar to that described in [2]. Agents communicate their current location to the Site they are currently registered with. Sites then engage in a client-server communication with the SiteServer, which aggregates all this information. In the next phase of the communication, the SiteServer returns this global information to each Site, which then passes it on to each Agent. Agents then act on this information and alter their current position. This is described in the code snippets below , and compares the three main processes of the simulation. The implementation of the pedestrian simulation is illustrated in Fig. 1.

3.1 Discover and Modify

In order to ensure that all processes are updated and modified in parallel, two Barriers are used: discover and modify. During the discover phase, all Sites are updated by the SiteServer with the global coordinates of every Agent. Each Site then updates all Agents that are registered with it.

As explained in [13], autonomous software agents perceive their environment and then act on it. This creates a two phase process for each step of the simulation, discovery and modification, that all processes comply with. These phases are enforced by barriers, described above. The tasks carried out by each type of process for each step of the simulation are described in the Table 1.

All communications between processes are on a client-server basis. In effect, a client-server relationship involves the client sending a request to the server, to which the server is guaranteed to respond [14] [15]. Processes at the same level do not communicate directly with each other, only with the process at the next level up. As stated in [16] 'such communication patterns have been proven to be deadlock free.'

3.2 Description of Space

The division of space between sites allows for a simulation that is scalable and robust, separating out the management of agents between many processes. The

Table 1. Processing Sequence

Agent	Site	SiteServer
Synchronise on discover barrier		
	Request global coordinates	→ Receive requests
	Receive global coordinates	← Send global coordinates
Request update →	Receive requests	
Receive update ←	Send global coordinates	
Synchronise on modify barrier		
Modify state		
Send state →	Receive state	
Receive ACK ←	Send ACK	
	Send updates	→ Receive updates
	Receive ACK	← Send ACK
		Aggregate updates into global coordinates

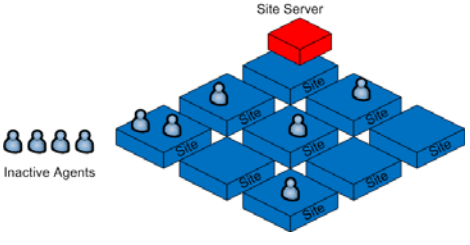


Fig. 1. Process layers

Site processes themselves have no knowledge of how they are situated. Each Agent class has a Map object that provides information about the area that a Site is associated with and the means with which the Agent can register with this site. In this way, Site processes can be associated with spaces of any shape or size. Theoretically, these spaces can range from triangles, the simplest co-planar two dimensional areas, up to complex three dimensional shapes.

At the edges of each space, an Agent may either migrate to the next Site, or encounter a hard boundary, requiring it to change direction. This is determined by the existence of a reference to the adjacent Site, if one exists. This is a reference to the Site's register channel, an Any2OneChannel, which allows many writers (the Agents seeking to register) and only one reader (the destination Site).

The register process happens in two phases. First the Agent must inform its current Site that it wishes to deregister, during the discovery phase. During the modify phase, before any other operation or communication is carried out, the Agent writes a reference to its communication channels to the register channel of the new Site, and waits for an acknowledgement. In this way, while an arbitrary number of Agents may wish to migrate from Site to Site at any one time, these attempts will always succeed.

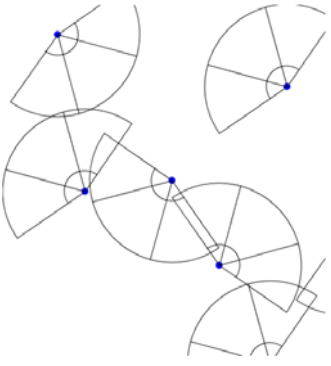


Fig. 2. Example application

4 Results

A number of test runs were performed to evaluate how the simulation performed when the number of agents was incremented. This was done in order to demonstrate the scalability of the system. This test was carried out with one Site object, and the number of agents incremented by ten for each run. The results are summarised in Table 2.

Table 2. Results from test runs

Number of agents	Total time	Avg time per step (ms)	Avg time per Agent (ms)
10	151	15.11	1.51
50	2057	41.13	0.82
100	7817	78.17	0.78
150	17454	116.36	0.78
200	31104	155.52	0.78

As can be seen from Table 2, the time to update each Agent during each step of the simulation is more or less constant. This is illustrated in Fig. 3 below.

These average times tend to decrease as the number of Agents increase. This reflects the overhead of setting up support processes, such as the display processes. Thereafter, the average times per Agent tend to settle at around 0.78 ms.

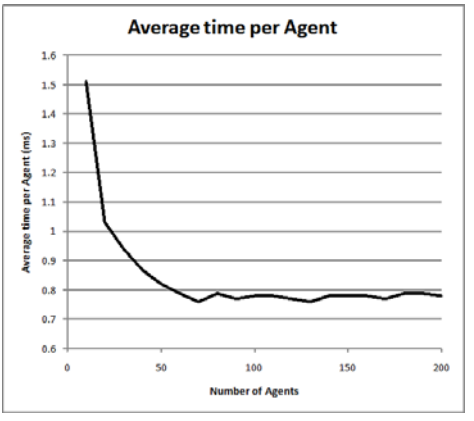


Fig. 3. Average update times per Agent (ms) by test run

5 Future Work

The current work implements simple reactive agents. These contain little in the way of intelligence in making their choices. Below is an image from software showing simple reactive Agents. Their field of view replicates that of humans. The span of human vision is 160 degrees. Central vision only occupies 60 degrees of this, with peripheral vision on each side occupying 50 degrees [17]. The minimum distance between agents is delimited by the inner arc of their field of view. Should any other Agent approach this, they will react by choosing a different direction.

Although many simple agents have been used to simulate emergent behaviour [18], the purpose of the TRAMP project is the simulation of human behaviour derived from data collected by infra-red sensors. As shown in Fig.4 actual human movements, when navigating across a space, are described by elegant and coherent curves. This is difficult to replicate using simple agents. In order to achieve this aim, agents trained using Learning Classifier Systems (LCS) will be developed, and their interactions studied.

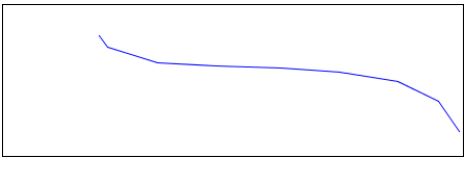


Fig. 4. Pedestrian trajectory recorded using infra-red sensors

6 Conclusion

In this paper, the application of CoSMoS design patterns in the development of MAS simulations has been discussed. The principles of concurrent processing using non-conventional techniques based on CSP and pi-calculus, that guarantee livelock and deadlock free concurrence, as well as eliminating race hazards have been explained. Other advantages of this approach, scalability and robustness, have also been demonstrated. This offers a firm foundation for future work, using MAS, to replicate pedestrian behaviour.

References

1. Dijkstra, J., Timmermans, H.J., Jessurun, A.: A Multi-Agent Cellular Automata System for Visualising Simulated pedestrian Activity. In: Bandini, S., Worsch, T. (eds.) Theoreteical and Practical Issues on Cellular Automata - Proceedings on the 4th International Conference on Cellular Automata for research and Industry, October 2000, pp. 29–36. Springer, Heidelberg (2000)
2. Quinn, M., Metoyer, R.A., Hunter-Zaworski, K.: Parallel implementation of the social forces model. Pedestrian and Evacuation Dynamics, 63–74 (2003)
3. Stepney, S., Welch, P., Timmis, J., Alexander, C., Barnes, F., Bates, M., Polack, F., Tyrrell, A.: CoSMoS: Complex Systems Modelling and Simulation infrastructure, EPSRC grants EP/E053505/1 and EP/E049419/1 (April 2007), http://www.cosmos-research.org/
4. Welch, P.H., Barnes, F., Polack, F.: Communicating complex systems. In: Hinchey, M.G. (ed.) Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS-2006), Stanford, California, August 2006, pp. 107–117. IEEE, Los Alamitos (2006)
5. Sampson, A., Welch, P.H., Barnes, F.: Lazy Cellular Automata with Communicating Processes. In: Broenink, J., Roebbers, H., Sunter, J., Welch, P.H., Wood, D.C. (eds.) Communicating Process Architectures 2005. Concurrent Systems Engineering Series, September 2005, vol. 63, pp. 165–175. IOS Press, The Netherlands (2005)
6. Welch, P.H., Vinter, B., Barnes, F.: Initial experiences with occam-pi simulations of blood clotting on the minimum intrusion grid. In: Arabnia, H.R. (ed.) Proceedings of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2005), Las Vegas, Nevada, USA, June 2005, pp. 201–207. CSREA Press (2005)
7. Andrews, P., Sampson, A., Bjrndalen, J.M., Stepney, S., Timmis, J., Warren, D., Welch, P.H., Noble, J.: Investigating patterns for the process-oriented modelling and simulation of space in complex systems. In: Bullock, S., Noble, J., Watson, R., Bedau, M.A. (eds.) Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems, Cambridge, MA, pp. 17–24. MIT Press, Cambridge (2008)
8. Hoare, C.A.R.: Communicating sequential processes. Prentice Hall, Englewood Cliffs (1985), ISBN 0 13 153271 5 (Hard), 0 13 153289 8 (Pbk)
9. Welch, P.H., Barnes, F.: Communicating mobile processes: introducing occam-pi. In: Abdallah, A.E., Jones, C.B., Sanders, J.W. (eds.) Communicating Sequential Processes. LNCS, vol. 3525, pp. 175–210. Springer, Heidelberg (2005)

10. Welch, P.H., Brown, N.C., Moores, J., Chalmers, K., Sputh, B.: Integrating and Extending JCSP. In: McEwan, A.A., Ifill, W., Welch, P.H. (eds.) Communicating Process Architectures 2007, pp. 349–369 (July 2007)
11. Hansen, P.B.: Java's insecure parallelism. SIGPLAN Not 34(4), 38–45 (1999)
12. McColl, W.F.: Scalable computing. In: Computer Science Today: Recent Trends and Developments, pp. 46–61. Springer, Heidelberg (1996)
13. Wooldridge, M.: An Introduction to MultiAgent Systems. John Wiley & Sons, Chichester (2002)
14. Welch, P.H., Justo, G., Willcock, C.: Higher-Level Paradigms for Deadlock-Free High-Performance Systems. In: Grebe, R., Hektor, J., Hilton, S., Jane, M., Welch, P.H. (eds.) Transputer Applications and Systems 1993, Proceedings of the 1993 World Transputer Congress, Aachen, Germany, September 1993, vol. 2, pp. 981–1004. IOS Press, Netherlands (1993)
15. Martin, J.M., Welch, P.H.: A Design Strategy for Deadlock-Free Concurrent Systems. Transputer Communications 3(4) (July 1997) (in Press)
16. Ritson, C.G., Welch, P.H.: A Process-Oriented Architecture for Complex System Modelling. In: McEwan, A.A., Ifill, W., Welch, P.H. (eds.) Communicating Process Architectures 2007, pp. 249–266 (July 2007)
17. Bruce, V., Green, P., Georgeson, M.: Visual Perception: Physiology, Psychology, and Ecology. Psychology Press, New York (1996)
18. Blue, V., Embrechts, M., Adler, J.: Cellular automata modeling of pedestrian movements. In: 1997 IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation, Orlando, FL, vol. 3, pp. 2320–2323 (1997)

Artificial Intelligence of Virtual People in CA FF Pedestrian Dynamics Model

Ekaterina Kirik[1,2], Tat'yana Yurgel'yan[2], and Dmitriy Krouglov[1,3]

[1] Institute of Computational Modelling of
Siberian Branch of Russian Academy of Sciences,
Krasnoyarsk, Akademgorodok, Russia, 660036
kirik@icm.krasn.ru
[2] Siberian Federal University, Krasnoyarsk, Russia
[3] V.N. Sukachev Institute of Forest of
Siberian Branch of Russian Academy of Sciences, Krasnoyarsk, Russia

Abstract. This paper deals with mathematical model of pedestrian flows. We focus here on an "intelligence" of virtual people. From macroscopic viewpoint pedestrian dynamics is already well simulated but from microscopic point of view typical features of people movement need to be implemented to models. At least such features are "keeping in mind" two strategies – the shortest path and the shortest time and keeping a certain distance from other people and obstacles if it is possible. In this paper we implement mathematical formalization of these features to stochastic cellular automata (CA) Floor Field (FF) model.

Keywords: cellular automata; pedestrian dynamics; transition probabilities.

1 Introduction

Modelling of pedestrian dynamics is actual problem at present days. Different approaches from the social force model ([1] and references therein) based on differential equations to stochastic CA models (for instance, [6,2,10] and references therein) are developed. They reproduce many collective, so to say, macroscopic properties including lane formation, oscillations of the direction at bottlenecks, the so-called "faster-is-slower" effect. These are an important and remarkable basis for pedestrian modelling. But there are still things to be done from microscopic point of view. The better individual pedestrian behavior is more realistic collective interaction and shape of flow are.

We have focused on a fact that in regular situations (non emergent) pedestrians choose their route more carefully [1]. And our aim is to mathematically formalize some features of individual behavior and as a result to improve simulation of individual and collective dynamics of people flow. We focus on implementation to a model such behavioral aspects of decision making process as: pedestrians keep a certain distance from other people and obstacles if it is possible; while moving people follow at least two strategies — the shortest path

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 513–520, 2010.

and the shortest time. Strategies may vary, cooperate and compete depending on current position.

Our model takes inspiration from stochastic floor field (FF) CA model [6] that provides pedestrians with a map which "shows" the shortest distance from current position to a target.

There were introduced [3] already some innovations ("people patience", "visibility radius", "environment analyzer") that allowed to extend basis FF model towards behavioral aspect and make more flexible/realistic decision making process. By this reason model obtained was named as *Intelligent FF CA model*. In this paper we extend previous result and develop further intelligence of virtual people in the sense mentioned above.

The article is organized as follows. In the next section, the problem is stated. In Section 3.1, we describe the model in general and update rules. Section 3.2 contains probability formulas. It is followed by the discussion and the presentation of the simulation results.

2 Statement of the Problem

The space (plane) is known and sampled into cells $40cm \times 40cm$ in size (it is an average space occupied by a pedestrian in a dense crowd [6]) which can either be empty or occupied by one pedestrian (particle) only:

$$f_{ij} = \begin{cases} 1, \text{ cell } (i,j) \text{ is occupied by a pedestrian;} \\ 0, \text{ cell } (i,j) \text{ is empty.} \end{cases}$$

Cells may be occupied by walls and other nonmovable obstacles:

$$w_{ij} = \begin{cases} 1, \text{ cell } (i,j) \text{ is occupied by an obstacle;} \\ 0, \text{ cell } (i,j) \text{ is empty.} \end{cases}$$

Starting people positions are known. Target point for each pedestrian is the nearest exit. Each particle from cell (i,j) can move to one of four its next-neighbor cells or to stay in the present cell (the von Neumann neighborhood $= \{(i-1,j),(i,j+1),(i+1,j),(i,j-1),(i,j)\}$) at each discrete time step $t \to t+1$, e.i., $v_{max} = 1$.

The direction to move for each particle at each time step is random and determined in accordance with transition probabilities distribution and rules.

Thus, to simulate the intelligent people movement (directed with typical features) the main task is to determine "right" transition probabilities and transition rules.

3 Solution

3.1 Update Rules

A typical scheme for stochastic CA models is used here. There is step of some preliminary calculations. Then at each time step transition probabilities are calculated and direction is chosen. If there are more then one candidates to one

cell a conflict resolution procedure is applied, and then simultaneous transition of all particles is made.

In our case, *preliminary step* includes calculations of static Floor Field (FF) S [6]. Field S coincides with the sampled space. Each cell $S_{i,j}$ saves shortest discreet distance from cell (i, j) to the nearest exit. It doesn't evolve with time and isn't changed by the presence of the particles. One can consider S as a map that pedestrians use to move to the nearest exit. While calculating field S we admit diagonal transitions and consider that vertical and horizontal movement to the nearest cell has a length of 1; length of diagonal movement to the nearest cell is $\sqrt{2}$. (And it's clear that movement through a corner of wall or collum is forbidden, and roundabout movement only is admitted in such cases.) It is made discreet distance more close to continuous one.

The probabilities to move from cell (i, j) to each of the four nearest cells are calculated in the following way:

$$p_{i-1,j} = \frac{\tilde{p}_{i-1,j}}{N_{i,j}}, p_{i,j+1} = \frac{\tilde{p}_{i,j+1}}{N_{i,j}}, p_{i+1,j} = \frac{\tilde{p}_{i+1,j}}{N_{i,j}}, p_{i,j-1} = \frac{\tilde{p}_{i,j-1}}{N_{i,j}}, \quad (1)$$

where $N_{i,j} = \tilde{p}_{i-1,j} + \tilde{p}_{i,j+1} + \tilde{p}_{i+1,j} + \tilde{p}_{i,j-1}$.

Moreover, $p_{i-1,j} = 0, p_{i,j+1} = 0, p_{i+1,j} = 0, p_{i,j-1} = 0$ only if $w_{i-1,j} = 1, w_{i,j+1} = 1, w_{i+1,j} = 1, w_{i,j-1} = 1$ correspondingly.

We don't calculate probability to stay the at present cell directly. But decision rules are organized in a way that such opportunity may be realized and people patience is reproduced by this means.

The decisions rules are [4]:

1. If $Norm_{i,j} = 0$, motion is forbidden; otherwise, a target cell $(l, m)^*$ is chosen randomly using the transition probabilities.
2. (a) If $Norm_{i,j} \neq 0$ and $(1 - f^*_{l,m}) = 1$, then target cell $(l, m)^*$ is fixed.
   (b) If $Norm_{i,j} \neq 0$ and $(1 - f^*_{l,m}) = 0$, then cell $(l, m)^*$ is not available for moving and "people patience" can be realized. For this purpose, the probabilities of cell $(l, m)^*$ and all the other occupied adjacent cells are given for the current position. Again, the target cell is chosen randomly using the transformed probability distribution[1].
3. Whenever two or more pedestrians have the same target cell, we use simple scheme to resolute conflicts [6][2]. Movement of all involved pedestrians is denied with probability μ, i.e., all pedestrians remain at their places. With probability $1 - \mu$ one of the candidates moves to the desired cell. The pedestrian that is allowed to move is chosen randomly.

[1] This trick of choosing the current position is provoked by the fact that when moving directionally people usually stop only if the preferable direction is occupied. The original FF model [9] never gives zero probability to the current position, and it may be chosen independent of the environment.

[2] Advanced method is reproduced here [10]. But present research doesn't concentrate on exit problems, we only need to provide flow path through narrow places.

4. The pedestrians allowed to move perform their motion to the target cell.
5. The pedestrians that stand in the exit cells are removed from the space.

These rules are applied to all the particles at the same time. , i.e., parallel update is used.

3.2 Probability

In stochastic CA pedestrian flow models the update rules preferably answer the question "How" to make movement. The transition probability preferably determines "Where" to move. To simulate the collective pedestrian movement with the realistic shape of flow one should use the "right" probability formulas. To make them "right" means to "remember" that in normal situations people choose their rout following some subconscious common rules (see [1] and reference therein). The rules are: a) pedestrians keep a certain distance from other people and obstacles, and more tight crowd this distance smaller, b) while moving people follow at least two strategies – the shortest path and the shortest time. Thus, mostly in this paper we focus on transition probabilities. And our aim is the attempt of mathematical formalization of the features mentioned above.

In FF models people move to the nearest exit and their wish to move there doesn't depend on current distance to exit. From probability view point this means that for each particle among all the nearest neighbor cells a neighbor with the smallest S should have the largest probability. So the main driving force for each pedestrian is to minimize FF S at each time step. But in this case, only the strategy of shortest path is mainly realized. This results in the fact that in the models people density does not regulate distance between people, and a slight regard to avoidance of congestions is supposed.

An idea to improve dynamics in FF model is to introduce environment analyzer in the probability formula. It should regulate distance between people and decrease the influence of the shortest path strategy and increase the possibility to move to a direction with favorable conditions for moving.

There were attempts already to introduce some environment analyzers into probabilities in stochastic CA models [5], [3]. A model presented in [3] reproduces individual dynamics in a proper way (pedestrian moved using more natural path and avoiding obstacles ahead). But collective dynamics was reproduced properly only in the certain rooms with simple geometry (room with one exit preferably in the middle of the wall; there were no turnings, bottlenecks, obstacles, etc.). A main reason of it was that the environment analyzer was not flexible and spatially adaptive and its weight was considerably less than the weight of the driving force.

In this paper we introduce revised idea of the environment analyzer [3] and make an attempt to mathematically formalize complex decision making process that people do choosing their path.

First, we present the transition probability formula; below, we will discuss it in detail. For example, the probability of movement from cell (i, j) to the upper neighbor is

$$\tilde{p}_{i-1,j} = A^{SFF}_{i-1,j} A^{people}_{i-1,j} A^{wall}_{i-1,j} (1 - w_{i-1,j}). \tag{2}$$

Here

- $A^{SFF}_{i-1,j} = \exp(k_S \triangle S_{i-1,j})$ is the main driven force:
  1. $\triangle S_{i-1,j} = S_{i,j} - S_{i-1,j}$;
  2. $k_S \geq 0$ is the (model) sensitivity parameter which can be interpreted as knowledge of the shortest way to the destination point or a wish to move to the destination point. The equality $k_S = 0$ means that the pedestrians ignore the information from field S and move randomly. The higher k_S, the better directed the movement.

  Since field S increases radially from the exit(s) in our model, then, $\triangle S_{i-1,j} > 0$ if cell $(i-1,j)$ is closer to the exit than the current cell (i,j), $\triangle S_{i-1,j} < 0$ if the current cell is closer, and $\triangle S_{i-1,j} = 0$ if cells (i,j) and $(i-1,j)$ are equidistant from the exit.

  In contrast to other authors that deal with FF model (e.g., [6], [2], [10]), we propose to use $\triangle S_{i-1,j}$. From mathematical view point, it is the same but computationally this trick has a great advantage. Values of FF may be too high (depending on the space size) and $\exp(k_S S_{i-1,j})$ may be uncomputable. This is a significant restriction. At the same time $0 \leq \triangle S_{i-1,j} \leq 1$, and problem of computing $A^{SFF}_{i-1,j}$ is absent;
- $A^{people}_{i-1,j} = \exp\left(-k_P D_{i-1,j}(r^*_{i-1,j})\right)$ is factor that takes into account people density in the given direction:
  1. $r^\star_{i-1,j}$ is a distance to the nearest obstacle in this direction ($r^\star_{i-1,j} \leq r$);
  2. $r > 0$ is the visibility radius (model parameter) representing the maximum distance (number of cells) at which the people density and the presence of obstacles influence on the probability in the given direction;
  3. the density lies within $0 \leq D_{i-1,j}(r^*_{i-1,j}) \leq 1$; if all the $r^*_{i-1,j}$ cells are empty in this direction, we have $D_{i-1,j}(r^*_{i-1,j}) = 0$; if all the $r^*_{i-1,j}$ cells are occupied by people in this direction, we have $D_{i-1,j}(r^*_{i-1,j}) = 1$. We estimate the density using the idea of the kernel Rosenblat-Parzen's density estimate ([8], [7]):

$$D_{i-1,j}(r^*_{i-1,j}) = \frac{\sum\limits_{m=1}^{r^*_{i-1,j}} \Phi\left(\frac{m}{C(r^*_{i-1,j})}\right) f_{i-m,j}}{r^*_{i-1,j}},$$

  were

$$\Phi(z) = \begin{cases} \left(0.335 - 0.067(z)^2\right) 4.4742, & |z| \leq \sqrt{5}; \\ 0; & |z| > \sqrt{5}, \end{cases} \tag{3}$$

  $C(r^*_{i-1,j}) = \frac{r^*_{i-1,j}+1}{\sqrt{5}}$;
  4. k_P is the (model) people sensitivity parameter which determines the effect of the people density. The higher parameter k_P, the more pronounced the shortest time strategy;
- $A^{wall}_{i-1,j} = \exp\left(-k_W(1 - \frac{r^*_{i-1,j}}{r})\tilde{1}(\triangle S_{i-1,j} - \max \triangle S_{i,j})\right)$ is the factor that takes into account walls and obstacle:

1. $k_W \geq k_S$ is the (model) wall sensitivity parameter which determines the effect of walls and obstacles;
2. $\max \triangle S_{i,j} = \max\{\triangle S_{i-1,j}, \triangle S_{i,j+1}, \triangle S_{i+1,j}, \triangle S_{i,j-1}\}$, $\tilde{1}(\phi) = \begin{cases} 0, & \phi < 0, \\ 1 & \text{otherwise.} \end{cases}$ An idea of the function $\tilde{1}(\triangle S_{i-1,j} - \max \triangle S_{i,j})$ comes from the fact that people avoid obstacles only when moving towards the destination point. When people make detours (in this case, field S is not minimized), approaching the obstacles is not excluded.

– NOTE that only walls and obstacles turn the transition probability to "zero".

The probabilities of movement from cell (i, j) to each of the four neighbors are:

$$p_{i-1,j} = N_{i,j}^{-1} \exp\big[k_S \triangle S_{i-1,j} - k_P D_{i-1,j}(r^*_{i-1,j}) - \\ - k_W (1 - \frac{r^*_{i-1,j}}{r}) \tilde{1}(\triangle S_{i-1,j} - \max \triangle S_{i,j})\big](1 - w_{i-1,j}); \quad (4)$$

$$p_{i,j+1} = N_{i,j}^{-1} \exp\big[k_S \triangle S_{i,j+1} - k_P D_{i,j+1}(r^*_{i,j+1}) - \\ k_W (1 - \frac{r^*_{i,j+1}}{r}) \tilde{1}(\triangle S_{i,j+1} - \max \triangle S_{i,j})\big](1 - w_{i,j+1}); \quad (5)$$

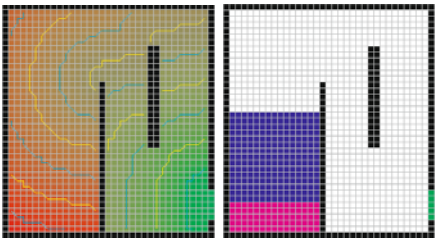
$$p_{i+1,j} = N_{i,j}^{-1} \exp\big[k_S \triangle S_{i+1,j} - k_P D_{i+1,j}(r^*_{i+1,j}) - \\ - k_W (1 - \frac{r^*_{i+1,j}}{r}) \tilde{1}(\triangle S_{i+1,j} - \max \triangle S_{i,j})\big](1 - w_{i+1,j}); \quad (6)$$

$$p_{i,j-1} = N_{i,j}^{-1} \exp\big[k_S \triangle S_{i,j-1} - k_P D_{i,j-1}(r^*_{i,j-1}) - \\ - k_W (1 - \frac{r^*_{i,j-1}}{r}) \tilde{1}(\triangle S_{i,j-1} - \max \triangle S_{i,j})\big](1 - w_{i,j-1}); \quad (7)$$

In expressions (4)-(7), the product $A^{people} A^{wall}$ is the environmental analyzer that deals with people and walls. The following restrictions take place $0 \leq \triangle S \leq 1$, $0 \leq D(r^*) \leq 1$, and $0 \leq 1 - \frac{r^*}{r} \leq 1$. These allows adjusting sensitivity of the model to the people density and the approaching to obstacles using parameters k_P and k_W, respectively. To be pronounced people and wall terms should not have parameters less then k_S ($k_P \geq k_S$, $k_W \geq k_S$).

Following the shortest time strategy means to take detour around high density regions if it is possible. Term A^{people} works as the reduction of the main driving force (that provides the shortest path strategy) and the probability of detours becomes higher. The higher k_P the more pronounced the shortest time strategy. Note that low people density makes influence of A^{people} small and probability of the shortest path strategy increases for particle. Parameters k_P allows to tune sensitivity of the model to people density.

Term A^{wall} corresponds only to avoidance of the ahead obstacles so it is not be discussed here. Assume that $k_W = k_S$.



a) Field S. b) Initial positions.

Fig. 1.

4 Simulations

Here we present some simulations result to demonstrate that our idea works. We use one space and compare 2 sets of parameters. Size of space is $14.8m \times 13.2m$ (37 cells $\times$ 33 cells) with one exit ($2.0m$). Static field S is presented in fig. 1a. In Fig. 1b are stating positions of the particles. They move towards the exit with the velocity $v = v_{max} = 1$.

Here we don't present some quantity results and only demonstrate quality difference of flow dynamics for 2 sets of model parameters for model presented.

The first set of parameters is $k_S = k_W = 4$, $k_P = 6$, $r = 10$. The second set is $k_S = k_W = 4$, $k_P = 18$, $r = 10$. Following moving condition are reproduced by both sets – pedestrians know way to exit very well, they want go to exit (it is determined by k_S), visibility is good (r), attitude to walls is "loyal" ($k_W = k_S$). Only parameter k_P varies here.

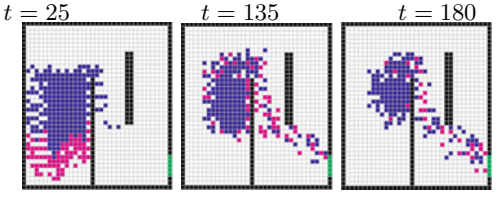


Fig. 2. Evacuation for 300 people, $k_S = k_W = 4$, $r = 10$, $k_P = 6$; $T_{tot} = 270[step]$

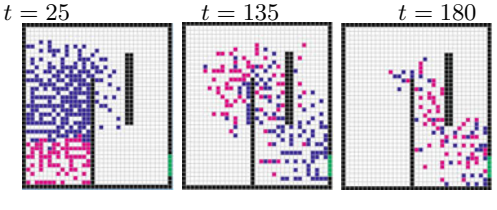


Fig. 3. Evacuation for 300 people, $k_S = k_W = 4$, $r = 10$, $k_P = 18$; $T_{tot} = 270[step]$

If $k_P = 6$ prevailing moving strategy is the shortest path (Fig. 2).

The other set of parameters $k_S = k_W = 4$, $k_P = 18$, $r = 10$ (see Fig. 3) allows to realize both strategies depending on conditions and regulate distance between people depending on density.

5 Conclusion

Under equal movement conditions $k_S = k_W = 4$, $r = 10$ different density sensitive parameters give significant divergence in the dynamics of the model. Combining of the shortest path and the shortest time strategies ($k_P = 18$) gives faster evacuation process, $T_{tot} = 270[step]$ (if $k_P = 6$, when the shortest path strategy predominates, $T_{tot} = 320[step]$), the higher turn radius, the using of detours facilities, the effective use of the exit width, and more realistic shape of flow in a whole. Model dynamics proper needs careful investigation and it is going on. Necessity of k_P spatial adaptation is already clear. It should be a function on space capacity.

References

1. Helbing, D.: Traffic related self-driven many-particle systems. Rev. Mod. Phys. 73(4), 1067–1141 (2001)
2. Henein, C.M., White, T.: Macroscopic effects of microscopic forces between agents in crowd models. Physica A 373, 694–718 (2007)
3. Kirik, E., Yurgel'yan, T., Krouglov, D.: An intelligent floor field cellular automation model for pedestrian dynamics. In: Proceedings of The Summer Computer Simulation Conference 2007. The Mission Valley Marriott San Diego, California, pp. 1031–1036 (2007)
4. Kirik, E., Yurgel'yan, T., Krouglov, D.: The Shortest Time and/or the Shortest Path Strategies in a CA FF Pedestrian Dynamics Model. Journal of Siberian Federal University, Mathematics and Physics 2(3), 271–278 (2009)
5. Malinetskiy, G.G., Stepantcov, M.E.: An application of cellular automation for people dynamics modelling. Journal of Computational Mathematics and Mathematical Physics 44(11), 2108–2112 (2004)
6. Nishinari, K., Kirchner, A., Namazi, A., Schadschneider, A.: Extended floor field CA model for evacuation dynamics. IEICE Trans. Inf. & Syst. E87-D, 726 (2004)
7. Parzen, E.: On estimation of probability Density Function. Ann. Math. Stat. 33, 1065–1076 (1962)
8. Rosenblat, M.: Remarks on some non-parametric estimates of a density function. Ann. Math. Stat. 27, 832–837 (1956)
9. Schadschneider, A., Seyfried, A.: Validation of CA models of pedestrian dynamics with fundamental diagrams. Cybernetics and Systems 40(5), 367–389 (2009)
10. Yanagisawa, D., Nishinari, K.: Mean-field theory for pedestrian outflow through an exit. Physical review E76, 061117 (2007)

Towards the Calibration of Pedestrian Stream Models

Wolfram Klein, Gerta Köster*, and Andreas Meister

Siemens AG, Germany
{gerta.koester,wolfram.klein}@siemens.com

Abstract. Every year people die at mass events when the crowd gets out of control. Urbanization and the increasing popularity of mass events, from soccer games to religious celebrations, enforce this trend. Thus, there is a strong need to gain better control over crowd behavior. Simulation of pedestrian streams can help to achieve this goal. In order to be useful, crowd simulations must correctly reproduce real crowd behavior. This usually depends on the actual situation and a number of socio-cultural parameters. In other words, what ever model we come up with, it must be calibrated. Fundamental diagrams capture a large number of the socio-cultural characteristics in a very simple concept. In this paper we represent a method to calibrate a pedestrian stream simulation tool so that it can reproduce arbitrary fundamental diagram with high accuracy. That is, it correctly reproduces a given dependency of pedestrian speed on the crowd density. We demonstrate the functionality of the method with a cellular automaton model.

Keywords: cellular automaton, pedestrian simulation, fundamental diagram, calibration.

1 Introduction

Crowd behavior has gained a lot of interest in the past years, fueled by the insight that mass events with their growing popularity represent a risk for civil security. Overall a comparably large variety of potentially dangerous scenarios is known. The scenarios range from environmental disasters to terrorist attacks. Each scenario comes with its own scale (building, housing block or city), cultural (e.g. India or Germany) [1] or event-specific [2] (e.g. sports game or religious celebration) characteristics.

To a large extend they all share the same quite general trait that in dense crowds that press towards a certain goal an individual can easily suffocate or be trampled to death. And of course there is always the need to evacuate people as fast as possible. Without being complete this illustrates the need to gain better control over crowd behavior.

Simulation of crowd streams can help to achieve this goal: Simulations allow running through a number of scenarios in a critical situation and thus to find

* Corresponding author.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 521–528, 2010.

adequate measures to improve security. In order to be useful, crowd simulations must correctly reproduce crowd behavior. Primarily, the model must capture the system dynamics, namely the most important mechanisms of interaction. Gradually increasing the details of modeling and comparing simulation results and empirical data has been established as a successful strategy for identifying these mechanisms. While the underlying mechanisms of interaction can be assumed to be quite universal, they depend on a large number of parameters. Especially rather basic or universal rules of interaction cannot be expected to capture every situation and need to be adapted to the scenario of interest. This is a vital issue for practical applications focusing on high-fidelity reproduction of a wide range of scenarios, each characterized by a corresponding set of parameters.

In other words, what ever model we come up with, it must be calibrated. Calibration, in our context, means that the model we use is adapted to specific information from the real world. This information can be extremely varied. Widely known dependencies are for example the basic environmental conditions like structural constraints imposed by the architecture of a surrounding building [3]. Recently also socio-cultural aspects have been investigated [1]. Obviously the range of possible parameters is large and the impact differs from scenario to scenario.

Hence, the first challenge of calibration is to decide where to begin. The present work proposes the use of the so-called fundamental diagram of pedestrian flow as the source of parameters capturing the most relevant parameters characterizing different scenarios. Originating from vehicular highway traffic [4], [5], the diagram describes the function relation between the number of cars on a road section and their velocity. In recent years fundamental diagrams have also been obtained for various other systems based on motile constituents [6], [7], [8].

Among these systems one also finds the empirical fundamental diagram of pedestrian dynamics [9], [6], [10]. The functional relation between the density of pedestrians and walking speed has been measured by several groups. For a detailed survey we refer to [9]. Overall we found clear indication that, indeed, a major number of parameters such as cultural differences are captured. Apparently 'the speed of Indian test persons is less dependent on density than the speed of German test persons'[1].

Besides these effects also a scenario dependence can be expected. In case of counterflowing streams of pedestrians or at a bottleneck, the average walking speed in one direction might also depend on the density of pedestrians moving in the opposite direction [11], [12].

From that it becomes clear, that there is no fundamental diagram that is true for every scenario, culture, location. Simulating crowds in a rail station in, say, Berlin should yield significantly different results from a rail station in Delhi. So ideally, before running a simulation, we would obtain a fundamental diagram – or a collection of such diagrams – that is suitable for the scenario we are interested in. Then we would calibrate our model, so that it reproduces the phenomenon expressed through the fundamental diagram. There are, of course, many more aspects worth investigating. Our choice is to start with the fundamental diagram.

In this paper we represent a method to calibrate a pedestrian stream simulation tool so that it can reproduce any given fundamental diagram. We demonstrate the functionality of the method with a cellular automaton model. Nevertheless, this approach might also be applied in a modified way to models of other classes.

2 A Glance at the Model

Our model of choice is a cellular automaton. This approach allows us to incorporate directly observable interaction in a very simple way. As we focus on application in real scenarios simulation speed, namely faster-than-real-time capability is a major issue. Experience from vehicular traffic [13] or pedestrian dynamics [6], [10] shows that these requirements are usually well met even for large systems. We investigate an area that may be bounded by walls and contain obstacles. Persons are leaving and entering through sources and sinks, namely entrances and exits.

The persons move in one single plane or several planes such as floors. So we may restrict ourselves to two dimensions. We cover the area of interest by cells. In principle, triangular, rectangular and hexagonal cells are possible [10], [14]. Although square cells seem to be the most popular choice, we prefer a hexagonal grid for its two additional 'natural' directions of movement compared to the square grid. Each cell, at each time step, has a state: It is either empty or occupied. A cell can be occupied by a single person. The cell size is chosen to accommodate an average sized European male (in light summer attire and without baggage). Sources and targets of pedestrians as well as obstacles also occupy cells.

The simulation dynamics themselves follow a specific kind of sequential update scheme. That is, the cells containing persons are updated in the order the persons have entered the scenario from a source.

The core of the model is contained in the 'automaton', that is, the set of rules according to which the cell states are updated when we step one ahead in time. For this, we burrow from physics namely electrodynamics. In principle pedestrians are treated as charged particles, say electrons. So pedestrians are attracted by positive charges, such as exits and repelled by negative charges such as other pedestrians or obstacles.

The forces between pedestrians, targets and obstacles are calculated through a potential field, using the properties of conservative force fields from physics, where the force can be expressed as the gradient of a suitable scalar function: the potential. In this, the model is very similar to any typical cellular automaton model based on potentials as, for example, described in [15], [16], [10], [17], [12], [18], [19] or in the web-published handbook of the TraffGo tool [20]. The pure electron based approach clearly has its limitations when modeling human behavior. For example, humans use what they see in front of them to coordinate their movement. There is no radial symmetry. Hence, our model enriches the basic ideas by a number of sub-models to compensate the most relevant shortcomings. Using the terminology in [10] our model is:

1. Microscopic.
2. Discrete.

3. Deterministic with stochastic aspects.
4. Rule based but potential driven.

In this paper we do not strive to give another complete description of the, very successful, cellular automaton approach based on potential fields. Nor do we intend to describe our particular choice of sub-models. Instead, we want to enhance any such model by an aspect that we think of the utmost importance for useful application: calibration. Hence, we will focus on those model parameters that the calibration algorithm needs.

The model parameter that we will use for calibration is the walking speed. It is directly accessible through experiments and measurements. Each person is generated with an individual speed that the person tries to achieve – and indeed does achieve when the path is free: the free flow velocity [13], [5] or, as we call it, the 'desired velocity'. The distribution of the speeds follows the suggestions in [9]. That is, we assume a normal distribution about some medium desired speed. Some persons wish to go faster, if given the chance, others are slower by habit. The different velocities are made possible by allowing a person to move forward multiple cells per simulation step.

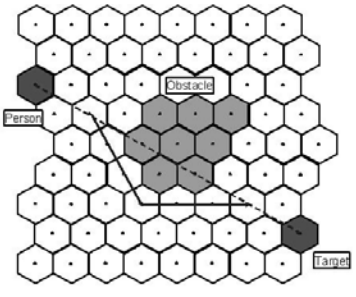


Fig. 1. A cellular automaton based on a grid with hexagonal cells. Pedestrians move from a source to a target avoiding obstacles on the way. We prefer a hexagonal grid over a square grid for its 2 additional 'natural'directions of movement.

3 The Challenge

The simulation results of our model without calibration show that it is suitable to qualitatively reproduce the fact that the denser the crowd, the slower the velocity of each person. That is, people are hindered by each other in their movement. However, in our simulations, they still move too fast compared to well known fundamental diagrams such as the popular diagram described by Weidmann [9]. See Figure 2 and Figure 3. This observation is – or was – shared by others simulation projects. See, for instance, the analysis according to the RIMEA guidelines conducted by [21]: test 4. In addition the simulated pedestrians appear to be 'short sighted'and do not decelerate before they literally 'bump' into a dense crowd.

Results are somewhat improved – at least when we only consider the fundamental diagram according to [9] – when the number of discrete speeds with which a person may move is increased. Please refer to Figure 2 and Figure 3. However, it is impossible to tune the basic model – e.g. by adjusting the repulsive force of indivuduals – so that it reproduces an arbitrary fundamental diagram with satisfactory accuracy on a quantitative level.

4 Solution Strategies

To meet the challenge of reproducing the given fundamental diagram quantitatively we introduce the new concept of deceleration classes. The basic idea is to measure the density of the crowd in the direction in which a person moves and then to adapt the person's velocity to the one suggested by the fundamental diagram of choice. The execution of the idea, however, is a little more complicated because the world of a cellular automaton is discretized through cells. Also, we do not want to loose individual differences in the human behavior.

Measuring the density means to count the number of persons in a reasonable number of cells that lie in the field of vision of a walking person. The field of vision is aligned with the direction in which the person walks. By this we reduce the basically two dimensional problem to the quasi one-dimensional situation for which fundamental diagrams are usually employed. As a result the pedestrians are no longer 'short sighted'but react to congestions ahead of them.

The model is first calibrated such that the mean free flow cell velocity – the average number of cells a person covers per simulation step when the path is free – corresponds to $1.34\frac{m}{s}$ as suggested in [9]. The cell velocities are normally distributed about this mean cell velocity (mcv). We get $2 \cdot mcv - 1$ discrete velocities.

When a person is surrounded by a dense crowd on the way to the chosen target, which lies in the direction of the lowest potential, the person's speed is adapted: More precisely, each person's desired velocity is temporarily reduced. Note that it still depends on the availability of a free path, whether the person can really achieve this new desired velocity. The number of cells by which the desired velocity of an individual is reduced for a certain density also depends on the original free flow velocity of the person, so that we maintain the individual differences in the walking speed. In our approach, the speed reduction depends on the density and free flow velocity. The dependency is expressed in a set of rules that involve artificial calibration parameters. Since the model's world is partitioned in cells the densities are discrete too. We therefore speak of density classes and, accordingly, of 'deceleration classes' to denote the calibration parameters that we introduced in the rules. Now, the deceleration classes must be tuned according to the fundamental diagram. This means, the model can be calibrated.

The impact of the calibrated deceleration classes on the simulation results is evident in Figure 4. We achieve an excellent fit with the fundamental diagram we used for calibration (taken from [9]).

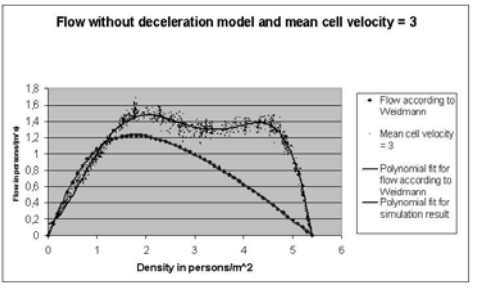


Fig. 2. Uncalibrated flow: without decleration model. The basic model does not correctly reproduce the dependency of the velocity on the density in a crowd.

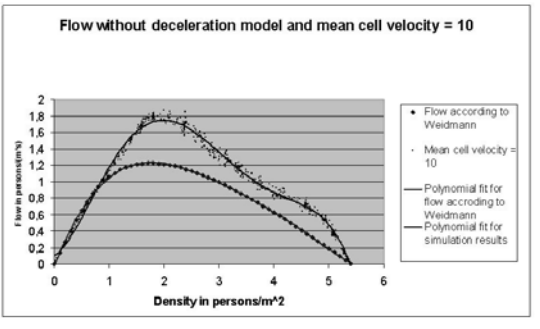


Fig. 3. Uncalibrated flow: without decleration model. The basic model does not correctly reproduce the dependency of the velocity on the density in a crowd. Increasing the number of possible velocities improves the results qualitatively, but does not allow to tune the model to an arbitrary fundamental diagram.

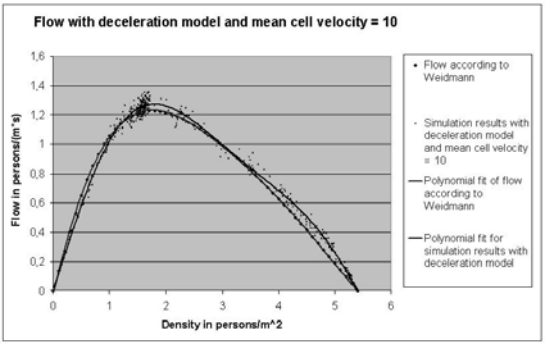


Fig. 4. Calibrated flow: with deceleration model. Introducing the deceleration model allows calibration. Parameters are tuned to an excellent fit with the fundamental diagram.

5 Conclusion

In this paper we discuss a first step towards the calibration of pedestrian stream models based on scenario specific fundamental diagrams. We consider the fundamental diagram as a behavioral model that aggregates a multitude of socio-cultural and even scenario dependent parameters. Ultimately, the differences in – say gender, nationality, fitness dependent on day time – find their expression in the way people walk as individuals and surrounded by a crowd. The walking speed is the crucial parameter. It depends on the density.

Thus, we do not need to identify each socio-cultural parameter of which, as a rule, we cannot quantify the influence anyway. Instead we feed the appropriate fundamental diagram into the simulation. In a learning phase, we adjust the way people slow down when they are walking in a crowd until the particular fundamental diagram is faithfully reproduced by computer simulations of our model.

To illustrate our calibration approach we have chosen a potential-based cellular automation model. We suggested so-called deceleration classes in this cellular automaton model to slow down persons when they approach a crowd on their way to a chosen target. We demonstrated that the calibrated deceleration classes are suitable to obtain a very good fit to a given fundamental diagram.

In an ideal set up, each scenario would have its own fundamental diagram. We therefore believe that the sort of calibration suggested here, is absolutely necessary to tune a simulation model to the scenario of interest. Clearly the number of measured fundamental diagrams currently available to the researcher is very limited. However, with new methods to gain data, such as video analysis or radio technologies, this deficiency may soon be overcome and it will become necessary to devise methods to automatically calibrate a simulation tool.

Robustness of our calibration, that is, sensitivity to variations and errors in the data is another vital issue. Furthermore, we will strive to refine and enlarge our method according the insight we expect from the increasing number of empirical fundamental diagrams.

We would like to thank Alexander John for useful discussions and pointing out some aspects regarding the presentation of the discussed material.

References

1. Chattaraj, U., Seyfried, A., Chakroborty, P.: Comparison of pedestrian fundamental diagram across cultures. arXiv:0903.0149, physics.soc-ph (2009)
2. Johansson, A., Helbing, D., A-Abideen, H.Z., Al-Bosta, S.: From crowd dynamics to crowd safety: A video-based analysis. Advances in Complex Systems 11(4), 497–527 (2008)
3. Predtetschenski, W., Milinski, A.: Personenströme in Gebäuden - Berechnungsmethoden für die Modellierung. Müller, Köln-Braunfeld (1971)
4. Greenshields, B.D.: A study of highway capacity. Proceedings Highway Research Record 14, 448–477 (1935)
5. May, A.D.: Traffic Flow Fundamentals. Prentice-Hall, Englewood Cliffs (1990)

6. Kretz, T.: Pedestrian Traffic. Simulation and Experimens. PhD thesis, Universität Duisburg-Essen (2007)
7. Chowdhury, D., Schadschneider, A., Nishinari, K.: Physics of transport and traffic phenomena in biology: from molecular motors and cells to organisms. Physics of Life Reviews 2, 318 (2005)
8. John, A., Schadschneider, A., Chowdhury, D., Nishinari, K.: Characteristics of ant-inspired traffic flow: Trafficlike collective movement of ants on trails: Absence of a jammed phase, http://arxiv.org/abs/0903.1434
9. Weidmann, U.: Transporttechnik für Fussgänger. Schriftenreihe des IVT 90 (1992)
10. Schadschneider, A., Klingsch, W., Klüpfel, H., Kretz, T., Rogsch, C., Seyfried, A.: Evacuation dynamics: Empirical results, modeling and applications, http://arxiv.org/abs/0802.1620
11. Kretz, T., Grünebohm, A., Schreckenberg, M.: Experimental study of pedestrian flow through a bottleneck. J. Stat. Mech (2006)
12. John, A., Schadschneider, A., Chowdhury, D., Nishinari, K.: Characteristics of ant-inspired traffic flow: Applying the social insect metaphor to traffic models. Swarm Intelligence 2 (2008)
13. Nagel, K., Schreckenberg, M.: A cellular automation model for freeway traffic. J. Phys. I France 2, 2221–2229 (1992)
14. Kinkeldey, C., Rose, M.: Fussgängersimulation auf der Basis sechseckiger zellularer Automaten, http://www.bauinf.uni-hannover.de/publikationen
15. Kluepfel, H.L.: A cellular automation model for crowd movement and egress simulation. PhD thesis, Universität Duisburg-Essen (2003)
16. Burstedde, C., Klauck, K., Schadschneider, A., Zittartz, J.: Simulation of pedestrian dynamics using a 2-dimensional cellular automation. Physika (Amsterdam) 295A, 507 (2001)
17. Hamacher, H.W., Tjandra, S.A.: Mathematical Modelling of Evacuation Problem: A State of the Art. Springer, Heidelberg (2002)
18. Kinkeldey, C.: Fussgängersimulation auf der Basis zellularer Automaten: Studienarbeit im Fach Bauinformatik (2003)
19. Rogsch, C.: Vergleichende Untersuchungen zur dynamischen Simulation von Personenströmen, Diplomarbeit an der Bergischen Universtät Wuppertal (2005)
20. TraffGo: Handbuch, http://www.traffgo-ht.com/de/pedestrians/downloads/index.html
21. Hebben, S.: Analyse RIMEA Projekt, TraffGo HT GmbH (2006), http://www.ped-net.org

Two Concurrent Algorithms of Discrete Potential Field Construction

Konrad Kułakowski and Jarosław Wąs

Institute of Automatics,
AGH University of Science and Technology
Al. Mickiewicza 30,
30-059 Cracow, Poland

Abstract. Increasing demand for computational power in contemporary constructions has created the need to build faster CPUs and construct more efficient algorithms. In this context especially the concurrent algorithms seem to be very promising. Depending on the number of available CPUs they may offer significant reductions in computation time.

In this article two concurrent algorithms of potential field generation are proposed. They present two different approaches to problem domain partitioning called by the authors respectively as *tearing* and *nibbling*. It is shown that depending on the problem topology either *Tear* algorithm or *Nibble* algorithm is faster. Conclusions are summed up in form of experimental results presenting how the algorithms work in practice. However algorithms construct a discrete potential field according to some specific scheme, there should be no major problems with generalization them to other potential field schemes.

1 Introduction

Potential field as a way of describing a space's local property is very popular in many different fields of science, such as physics, chemistry or mathematics. In computer science and automatics potential field is very often used in robot path planning [1,2,3,4], navigation and simulation of complex phenomena, such as pedestrian dynamics [5,6,7]. Depending on their specific application, potential fields may have different representations and construction algorithms. One possible form of discrete potential field is a regular grid, where the value of potential is assigned to every cell of a space [8,9]. Such a grid represents the occupancy of a space *(occupancy grid)* in which every cell may have several states corresponding to different types of objects that may occupy the cell (figure 1). Every cell has a value of potential assigned to it. Thus, every object in a space may find a collision-free way to potential sources (points of interest) by following the increasing or decreasing potential values. This property is often used in different navigation algorithms using potential field [9,10]. In static models, potential field is constructed once, at the beginning of path planning or simulation. In dynamically changing environments, potential field has to be recalculated every time the environment is changed. In such cases the time indispensable for re-construction

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 529–538, 2010.

of potential field may significantly impact on the total time of simulation, step of path planning routine etc. For this reason it is important to work on practical and efficient algorithms of potential field construction.

In this paper two such algorithms are presented. They are designed to be run on MIMD *(Multiple Instructions, Multiple Data)* architecture with shared memory [11]. Both algorithms minimize the computation time in comparison to their sequential versions. The obtained speedup depends on the space topology and might be different for each algorithm. Experimental results are also presented.

Both algorithms are presented using "object-oriented pseudo-code". Although the authors hope it is intuitive, it seems that some syntactic elements need to be explained in more detail. Thus, all the beginnings and endings of classes and methods are denoted by phrase *class*, *end class*, *method*, *end method*. Blocks of code are written using the same indentation depth. The assignment operator is denoted with the help of the sign ← and the method call on an object is denoted by "." operator.

2 Potential Field Construction

For the purpose of this paper a rather simple function describing the distribution of potential has been chosen. The space for which the potential field is defined is represented by an occupancy grid with square cells. Every cell may represent an obstacle, an empty space, or a source of potential (e.g. a door in the case of indoor pedestrian dynamics). The Moore neighbourhood defined for the cell c is a set of eight cells where each has at least one point of its border common with the border of c. The Moore neighbourhood of c will be denoted by $M(c)$. So, the potential function f_p for the cell c is defined as follows:

$$f_p(c) = \begin{cases} 0 & \iff c \text{ represents a potential source} \\ 1 + \min_{k \in M(c)}(f_p(k)) & \iff c \text{ represents an empty space} \\ \text{not defined} & \iff c \text{ represents an obstacle} \end{cases} \quad (1)$$

In the figure 1 we can see the hypothetical space with two potential sources (brighter squares on the left and upper edge) with the f_p calculated for every cell of the space which is not an obstacle (black cells).

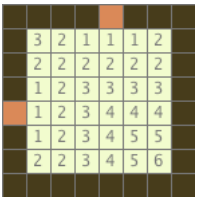


Fig. 1. Distribution of f_p for the small room (space) with two doors (two potential sources)

It is worth noticing that the calculation of $f_p(c)$ is local, which means that only a few other cells' f_p (to be more precise their Moore neighbourhood) have to be known to estimate $f_p(c)$. This fact enables parallelization of the whole potential field calculation process. Since many other potential construction algorithms require only local information to compute the value of potential in the given place, this fact should also allow the presented solutions to be generalized to the other types of potential fields e.g. using different definitions of distance [5] etc. Another benefit of f_p is that the potential field described by this function is local minima free and trap free.

2.1 Class Task

The common part of both the *Tear* and *Nibble* algorithms is a class *Task* (listing 1). This constructs the value of potential for some cells following the scheme given by the function f_p. The class *Task* has two fields: *row* and *nextRow*. The whole calculation take places in the function *makeStep()*.

Code listing 1: Task class responsible for local potential field calculation

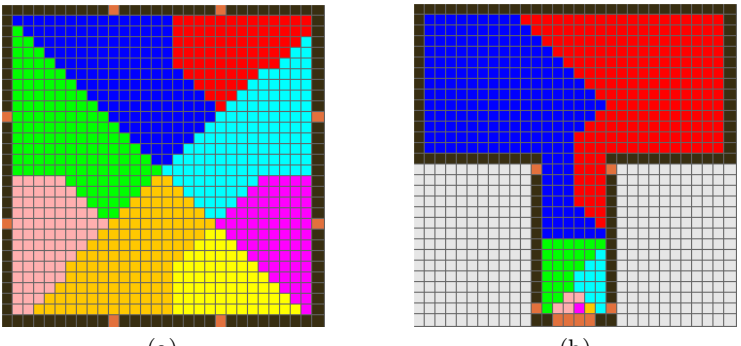
```
1  class Task
2    row
3    nextRow
4    method Task(Cell source)
5      row.add(source)
6    end method
7    method makeStep()
8      foreach c in row do
9        mr ← getMooreNeighbourhood(c)
10       foreach m in mr do
11         lock the access to m and c
12           if c.potential + 1 < m.potential then
13             m.potential ← c.potential + 1
14             nextRow.add(m);
15         unlock the access to cells m and c
16     row ← nextRow
17     makeEmpty(nextRow)
18   end method
19   method isNotFinished()
20     return row.isNotEmpty()
21   end method
22 end class
```

At the very beginning of this method the *row* contains cells for which the value of potential is known, whilst the *nextRow* is empty. While browsing the Moore neighbourhoods for every cell c coming from *row* (line 10)the potential value for every neighbour of c is calculated (lines 12 - 13). Every cell which

gets a new value of potential is added to *nextRow* (line 14). This is because the cell may impact on the potential values of its neighbours, so it should be taken into further consideration (line 16). Different tasks may be executed in different concurrent threads. Because cells are the resources shared between tasks every read and write value of potential from a cell has to be protected by a monitor. In the class *Task* the synchronization block simultaneously protects access to the two variables c and m (lines 11 - 15). In the case of c the monitor protects the read access whilst in the case of m it protects read and write access. The task is considered as finished if there are no cells for further processing in the *row*. In such a case the second call of *makeStep()* over the given worker is not necessary. Thus the typical usage of instance of the class *Tasks* consists of initialization of the *row* (by e.g. a potential source) and successive calls of the methods: *makeStep()* and *isNotFinished()*.

2.2 Tear Algorithm

The algorithm implemented by the class *Task* (listing 1) allows for computation of potential field generated by a single potential source. Thus, calculation of potential field with respect to many potential sources requires subsequent calls of *makeStep()* over all the tasks associated with these sources. Since different workers may have different sets of tasks, the whole calculation can be done concurrently. The *Tear* algorithm (listing 2) is just an implementation of this concurrent computation scheme. The *Tear* algorithm starts with initialization of variables storing potential sources and workers (lines 24, 25). It is assumed that there are as many task objects as potential sources and as many workers as logical processors (hereafter referred to as CPUs) in the system.



(a) (b)

Fig. 2. Workers' distribution in *Tear* algorithms for two different topologies. (a) a square space (30×30 cells) with eight potential sources and eight workers and, (b) a space with a corridor where six potential sources are placed closely to each other at the end of the corridor, and two potential sources are placed at the beginning of the corridor.

Code listing 2: Tear concurrent potenential field generation algorithm

```
23 method TearPotentialFieldMaker
24   sources ← initPotSources()
25   workers ← initWorkers()
26   foreach s in sources do
27     w ← chooseFrom(workers)
28     t ← Task(s)
29     w.add(t)
30   parfor w in workers do
31     w.go()
32 end method

33 class Worker {
34   Queue taskQueue

35   method add(task)
36     taskQueue.add(task)
37   end method

38   method go() {
39     repeat
40       Task t ← taskQueue.poll()
41       t.makeStep()
42       if t.isNotFinished() then
43         taskQueue.add(t)
44     until taskQueue.isEmpty()
45   end method

46 end class
```

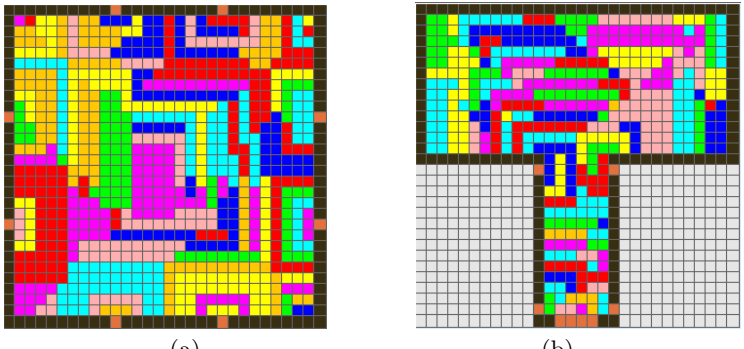
Next, the tasks are evenly assigned to the workers (lines 27, 29), then all the workers start concurrently (line 30). Since there might be more tasks than workers (usually there are many potential sources but only a limited number of CPUs) a single worker has to be able to handle more than one task. Thus the core functionality of the class *Worker* relies on queuing tasks (lines 36, 43) and subsequent calculation steps over the queued tasks (line 41). Recurrent calling of *makeStep()* over the given task lasts as long as the task is not finished. Observing how workers calculate the value of potential for different cells, some may think that every worker tries to tear for itself as much of the space as possible. In figure 2 we can see the result of *Tear* applied to two spaces of different shapes. Distinct colors (shades of grey) represent distinct workers. The difference in the cell count computed by different threads comes from the fact that for test purposes the *parfor* command has been implemented as an ordinary *for* command and workers, in fact, are started subsequently in loop. In such a small space as 30×30 cells the order of running workers is important, however, in larger spaces it does not matter. The strength of the *Tear* algorithm relies on its simplicity. It is a natural concurrent generalization of a sequential algorithm generating potential field according to the scheme given by the function f_p. It uses only one synchronization block connected directly with updating the potential value in

cells. This fact, connected with the algorithms' simplicity makes them reasonably fast and easy to implement. Permanent assignment of a potential source to a task and a task to a worker makes the algorithm simple, but it also means they are not flexible enough in spaces with more complex topologies. For instance, workers that have tasks holding potential sources actually placed close together are able to tear less space than workers with tasks having potential sources located far from each other. An example of such a situation is presented in the figure 2b. Here it might be observed how two workers responsible for two different potential sources "close" the exit of the corridor and do not allow the other workers to reach the main part of the space. As a result two workers out of eight do over eighty percent of the indispensable calculation, which is of course not optimum. For this reason more complex topologies require an algorithm which is able to allocate the computations more evenly.

2.3 Nibble Algorithm

Whilst the *Tear* algorithm might be not efficient in some specific cases, the new *Nibble* algorithm (listing 3) tries to overcome this problem. In the new algorithm the assignment task to a worker is not fixed and it might change once per step. Moreover it is possible to split a task into several sub-tasks if the task is too large. As a result, one worker may calculate values of potential in many different parts of the space. It looks like the worker would like to nibble some cell here, a little bit there (figure 3a and 3b).

At the beginning, the *Nibble* algorithm (similarly to *Tear* one) prepares potential sources and workers (lines 48, 49). Next, (line 50) the *WorkerManager* object is initialized. This object is responsible for workers' pooling and assigning tasks to workers. Since its functionality agrees with a thread pool pattern known from literature [12] it is not discussed here in detail. In the next two lines the first group of tasks are scheduled for execution. Every task is initialized by a single potential source. The most significant difference between *Tear* and *Nibble*



(a) (b)

Fig. 3. Workers' distribution in a *Nibble* algorithm applied to (a) a square space, (b) a space with a corridor

algorithms might be observed in the *Worker's* method *go()* (lines 59 – 67). First (line 60), it is checked whether the given task is not to large (i.e. whether the list of cells intended to process in the current step) is not larger than the sum of the size of all the processed tasks divided by the number of workers. If so, the current task (line 2) is split into an appropriate number of sub-tasks (Task splitting relies on dividing the list variable *row* into sub-lists). Next, all the sub-tasks but one are scheduled for execution by the *WorkerManager*. Then, the method *makeStep()* of the left task is called (line 64). Afterwards it is checked whether the task is not finished (line 65) and if so, the task is scheduled for further processing (line 66).

Code listing 3: Nibble concurrent potenential field construction algorithm

```
47 method NibblePotentialFieldMaker
48   sources ← initPotentialSources()
49   workers ← initWorkers()
50   workerManager ← initWorkerManager(workers)
51   foreach s in source do
52     workerManager.scheduleToExecution(Task(s))
53 end method

54 class Worker {
55   Task task
56   method Worker(t)
57     task ← t
58   end method
59   method go() {
60       if size(task) > optimalSize(task) then
61         split the 'task' to several subtasks t_1, ..., t_k so that every t_i has an optimal
   size
62         schedule tasks t_2, ..., t_k to execution
63         task← t_1
64       task.makeStep()
65       if task.isNotFinished() then
66         workerManager.scheduleToExecution(task)
67   end method
68 end class
```

Because in the *Nibble* algorithm tasks are not permanently associated with workers (but the assignment may change between steps), the situation that some workers throttle the others will never happen again. Moreover, the ability to split task into sub-tasks ensures that the algorithm always tries to maximise the usage of workers. As a result the *Nibble* algorithm works well even if there are some irregularities in the topology of the space or in the distribution of potential sources.

3 Experimental Results

Both algorithms have been implemented and tested for potential fields of different size and topology (figures 2a, 2b). As an implementation platform Java Standard Edition 6 was used. The performance tests were carried out on a server box equipped with four dual core processors Intel Xeon E5310 and 3GB of RAM. The concurrent implementations of the *Tear* and *Nibble* algorithms written in Java use standard objects and interfaces available in the Java package *java.util.concurrent*. For instance, classes such as *Task* and *Worker* are implemented as instances of the interface *Runnable*. *WorkerManager* was implemented as a *ThreadPoolExecutor* equipped with customized *ThreadFactory*. During tests all the CPUs were available for the JVM *(Java Virtual Machine)*. From the observation of CPU usage it appears that JVM tries to maximize utilization of all the available CPUs. Because Java classes are loaded into JVM on demand and this property of Java may forge the performance results (it may introduce super linear speedup), every series of tests has been preceded by JVM warm-up. During the warm-up phase the tested algorithm was run a couple of times so that the run time of the algorithm stabilized around some value. For both algorithms the appropriate charts showing dependency between the number of workers and execution time are presented (figure: 4). Besides the performance the speed up of algorithms is also shown on charts (figure 5). It is calculated as the quotient T_1/T_p where T_p is the execution time with the use of p workers.

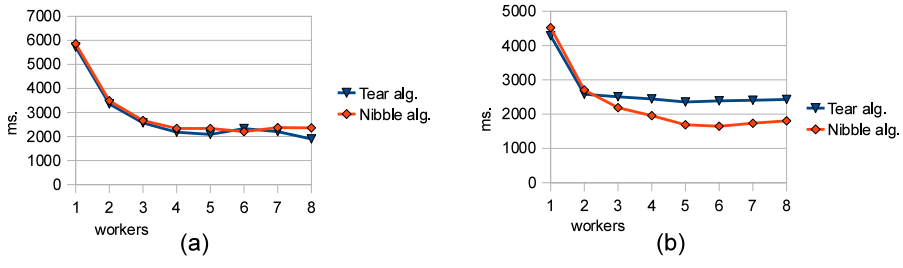


Fig. 4. Execution time of the *Tear* and *Nibble* algorithms with (a) a square room and (b) a room with a corridor

The conducted tests show that for a space with simple topology the multi-threaded version of the *Tear* algorithm is able to run almost three times faster than a single-thread algorithm. The multi-threaded *Nibble* algorithm is able to run two and a half times faster than its single-threaded version (figure: 5b). The speedup is not linear. This is because of many different reasons where the most important seem to be as follows:

- increase in synchronization overhead along the increase in number of workers,
- demand for resources from Java Virtual Machine (garbage collecting, memory allocation),

– property of the algorithms that the value of potential for the given cell might be calculated more than once.

The first two of the above are conclusions from the profiling sessions. The last one is a feature of both algorithms and is connected with the fact that for some cells the value of potential is calculated more than once by different threads. Fortunately, tests prove, that the overhead connected with the last issue is not so huge, and only about $3 - 5\%$ of cells are updated more than once. All the tests were carried out for the grid size 1000×1000 cells.

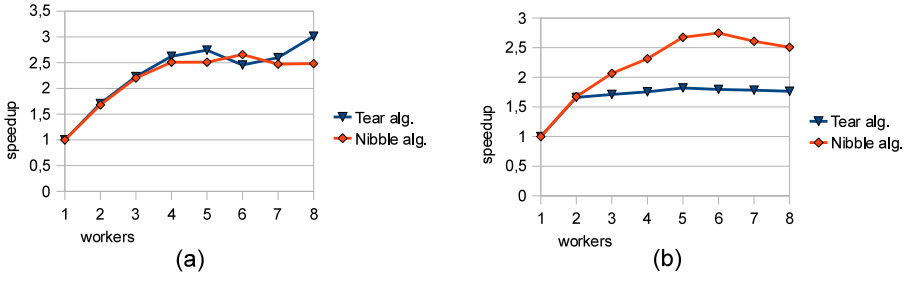


Fig. 5. Speedup of the *Tear* and *Nibble* algorithms with (a) a square room and (b) a room with a corridor

4 Summary

In this paper the two concurrent algorithms of discrete potential field calculation have been presented. Both, the *Tear* and *Nibble* algorithms are able to accelerate potential field calculation on a multi-core machine, however, depending on space topology and potential source deployment both algorithms, might sometimes be better. The preliminary tests prove that they offer up to threefold speedup using five – six processors. The experiments have been conducted on an ordinary Linux based multi-core computer and the whole solution has been implemented using freely available technologies.

Although the algorithms have been provisionally tested and tuned, it seems that there is still some room for further improvement. Some questions have also not yet been answered. For instance, the question about the efficiency of these algorithms in other software and hardware platforms is still open. For this reason, further development of these algorithms, besides the theoretical research, will also be focused on implementation issues. The authors hope that work in this subject brings further interesting and valuable results in future.

Acknowledgment

This research is financed by the Polish Ministry of Education and Science, project no.: N N516 228735 and by AGH University of Science and Technology, contract no.: 10.10.120.105.

References

1. Geraerts, R., Overmars, M.H.: The Corridor Map Method: Real-Time High-Quality Path Planning. In: Proceedings of International Conference on Robotics and Automation, ICRA (2007)
2. Heinemann, P., Becker, H., Zell, A.: Improved path planning in highly dynamic environments based on time variant potential fields. VDI BERICHTE (2006)
3. Wang, Y., Chirikjian, G.S.: A new potential field method for robot path planning. In: Proceedings of International Conference on Robotics and Automation (2000)
4. Barraquand, J., Langlas, B., Latombe, J.C.: Numerical potential field techniques for robot path planning. IEEE Trans. Systems, Man and Cybernetics (1992)
5. Kretz, T., Bonisch, C., Vortisch, P.: Comparison of various methods for the calculation of the distance potential field (2008)
6. Dudek-Dyduch, E., Wąs, J.: Knowledge representation of Pedestrian Dynamics in Crowd. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (eds.) ICAISC 2006. LNCS (LNAI), vol. 4029, pp. 1101–1110. Springer, Heidelberg (2006)
7. Treuille, A., Cooper, S., Popović, Z.: Continuum crowds. In: Proceedings of International Conference on Computer Graphics and Interactive Techniques (2006)
8. Murphy, R.R.: Introduction to AI Robotics. MIT Press, Cambridge (2000)
9. Behring, C., Bracho, M., Castro, M., Moreno, J.A.: An algorithm for robot path planning with cellular automata. In: Proceedings of the Fourth International Conference on Cellular Automata for Research and Industry (2000)
10. Xiaoxi, H., Leiting, C.: Path planning based on grid-potential fields. In: Proceedings of International Conference on Computer Science and Software Engineering (2008)
11. Flynn, M.: Some computer organizations and their effectiveness. IEEE Transactions on Computers (1972)
12. Lea, D.: Concurrent Programming in Java. In: Design Principles and Patterns, 2nd edn. Addison-Wesley Longman Publishing Co., Inc., Amsterdam (1999)

Frustration and Collectivity in Spatial Networks

Anna Mańka-Krasoń and Krzysztof Kułakowski

Faculty of Physics and Applied Computer Science, AGH University of Science and Technology, al. Mickiewicza 30, PL-30059 Kraków, Poland
impresja@gmail.com, kulakowski@novell.ftj.agh.edu.pl

Abstract. In random networks decorated with Ising spins, an increase of the density of frustrations reduces the transition temperature of the spin-glass ordering. This result is in contradiction to the Bethe theory. Here we investigate if this effect depends on the small-world property of the network. The results on the specific heat and the spin susceptibility indicate that the effect appears also in spatial networks.

Keywords: spatial networks; spin-glass.

1 Introduction

A random network is an archetypal example of a complex system [1]. If we decorate the network nodes with some additional variables, the problem can be mapped to several applications. In the simplest case, these variables are two-valued; these can be sex or opinion (yes or no) in social networks, states ON and OFF in genetic networks, 'sell' and 'buy' in trade networks and so on. Information on stationary states of one such system can be useful for the whole class of problems in various areas of science. The subject of this text is the antiferromagnetic network, where the variables are Ising spins $S_i = \pm 1/2$. As it was discussed in [1], the ground state problem of this network can be mapped onto the MAX-CUT problem, which belongs to the class of NP-complete optimization problems. Also, the state of the antiferromagnetic network in a weak magnetic field gives an information on the minimal vertex cover of the network, which is another famous NP-complete problem [1]. Further, in the ground state of the antiferromagnetic network all neighboring spins should be antiparallel, i.e. their product should be equal to -1. This can be seen as an equivalent to the problem of satisfiability of K conditions imposed on N variables, where N is the number of nodes and K is the number of links. The problem of satisfiability is known also to be NP-complete [2]. Here we are particularly interested in an ifluence of the network topology on the collective magnetic state of the Ising network. The topology is to be characterized by the clustering coefficient C, which is a measure of the density of triads of linked nodes in the network. In antiferromagnetic systems, these triads contribute to the ground state energy, because three neighboring spins of a triad cannot be antiparallel simultaneously to each other. This effect is known as spin frustration. When the frustration is combined with the topological disorder of a random network, the ground state of the system

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 539–546, 2010.

is expected to be the spin-glass state, a most mysterious magnetic phase which remains under dispute for more than thirty years [3,4,5]. These facts suggest that a search on random network with Ising spins and antiferromagnetic interaction can be worthwhile.

Here we are interested in an influence of the density of frustration on the phase transition from the disordered paramagnetic phase to the ordered spin-glass phase. In our Ising systems, the interaction is short-ranged and the dynamics is ruled by a simple Monte-Carlo heat-bath algorithm, with one parameter J/k_BT, i.e. the ratio of the exchange energy constant J to the thermal energy k_BT [6]. Despite this simplicity, the low-temperature phase is very complex and multi-degenerate even in periodic systems, where the topological disorder is absent [7]. Current theory of Ising spin-glasses in random networks ignores the contribution of frustrations, reducing the network to a tree [1]. In a 'tree-like structure' closed loops as triads are absent. In the case of trees the Bethe theory is known to work well [1,8]. In our considerations, the Bethe formula for the transition temperature T_{SG} from the paramagnetic to the spin glass phase [1]

$$\frac{-2J}{T_{SG}} = \ln \frac{\sqrt{B}+1}{\sqrt{B}-1} \tag{1}$$

serves as a reference case without frustrations. Here $B = z_2/z_1$ is the ratio of the mean number of second neighbours to the mean number of the first neighbours. Then, the transition temperature T_{SG} depends on the network topology. We note that in our network there is no bond disorder; all interactions are antiferromagnetic [9].

In our former texts, we calculated the transition temperature T_{SG} of the Erdös-Rényi networks [10] and of the regular network [11]. The results indicated that on the contrary to the anticipations of the Bethe theory T_{SG} decreases with the clustering coefficient C. However, in both cases we dealt with the networks endowed with the small-world property. It is not clear what dimension should be assigned to these networks, but it is well known that the dimensionality and in general the network topology influences the values of temperatures of phase transitions [12,8,13]. On the other hand, many real networks are embedded in the three-dimensional space - these are called spatial networks [14]. In particular, magnetic systems belong obviously to this class. Therefore, the aim of this work is to calculate the phase transition temperature T_{SG} again for the spatial networks. As in our previous texts [10,11] the clustering coefficient C is varied as to investigate the influence of the density of frustrations on T_{SG}.

In the next section we describe the calculation scheme, including the details on the control of the clustering coefficient. Third section is devoted to our numerical results. These are the thermal dependences of the magnetic susceptibility $\chi(T)$ and of the spacific heat $C_v(T)$. Final conclusions are given in the last section.

2 Calculations

The spatial network is constructed as follows. Three coordinates of the positions of nodes are selected randomly from the homogeneous distribution between 0

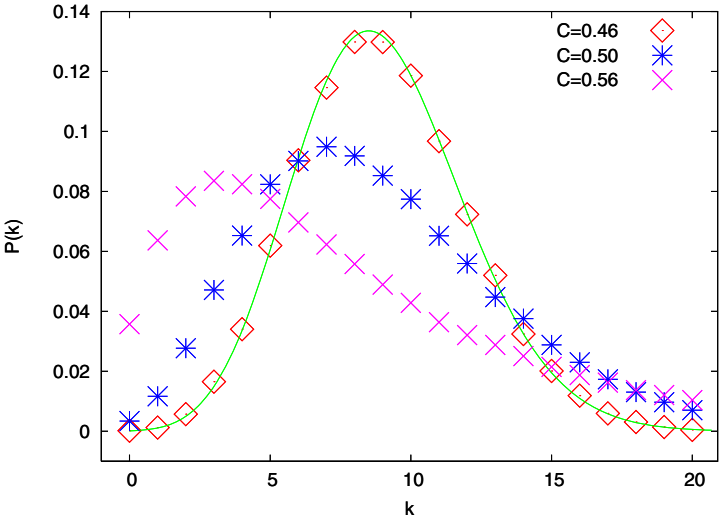


Fig. 1. The degree distribution for the mean degree $< k >= 9$ and three different values of the clustering coefficient C

and 1. Two nodes are linked if their mutual distance is not larger than some critical value a. In this way a controls the mean number of neighbours, i.e. the mean degree $< k >$ of the network. In networks obtained in this way, the degree distribution $P(k)$ agrees with the Poisson curve. As a rule, the number of nodes $N = 10^5$. Then, to obtain $< k >= 4$ and $< k >= 9$ we use $a = 0.0212$ and $a = 0.0278$. The mean degree $< k >$ is found to be proportional to $a^{2.91}$. This departure from the cubic function is due to the open boundary conditions. In two above cases, the values of the clustering coefficient C are respectively 0.42 and 0.47.

Now we intend to produce spatial networks with given mean degree $< k >$ and with enhanced clusterization coefficient C. This is done in two steps. First we adjust the radius a to obtain a smaller $< k >$, than desired. Next we apply the procedure proposed by Holme and Kim [15]: for each pair of neighbours of the same node a link between these neighbours is added with a given probability p'. This p' is tuned as to obtain a desired value of the mean degree $< k >$. Simultaneously, the clustering coefficient C is higher. In this way we obtain C between 0.42 and 0.46 for $< k >= 4$, and between 0.47 and 0.56 for $< k >= 9$. The degree distribution $P(k)$ in the network with enhanced C differs from the Poisson distribution, as shown in Fig. 1.

The dynamics of the system is ruled by the standard Monte Carlo heat-bath algorithm [6]. We checked that for temperature $T > 0.5$, the system equilibrates after 10^3 Monte Carlo steps; in one step each spin is checked. Sample runs ensured that after this time, the specific heat C_v calculations from the thermal derivative of energy and from the energy fluctuations give - within the numerical accuracy - the same results.

3 Results

In Fig. 2 we show the thermal dependence of the static magnetic susceptibility $\chi(T)$ for the network with mean degree $< k >= 4$. Fig. 3 displays the magnetic

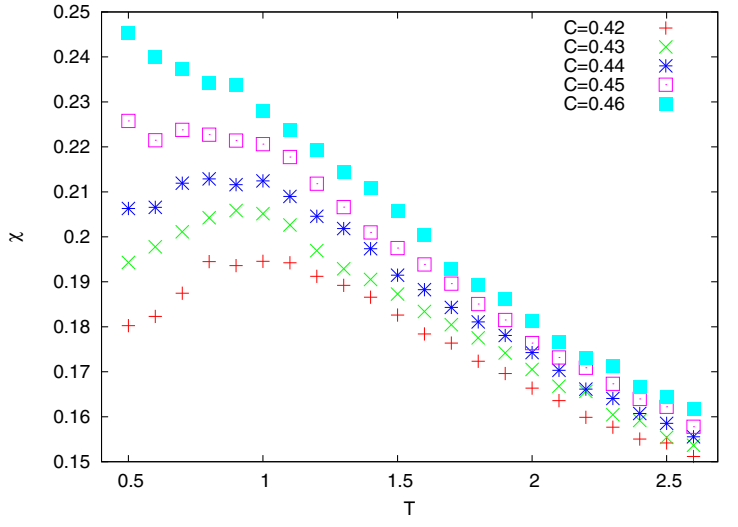


Fig. 2. The magnetic susceptibility $\chi(T)$ for $< k >= 4$ and different values of the clustering coefficient C

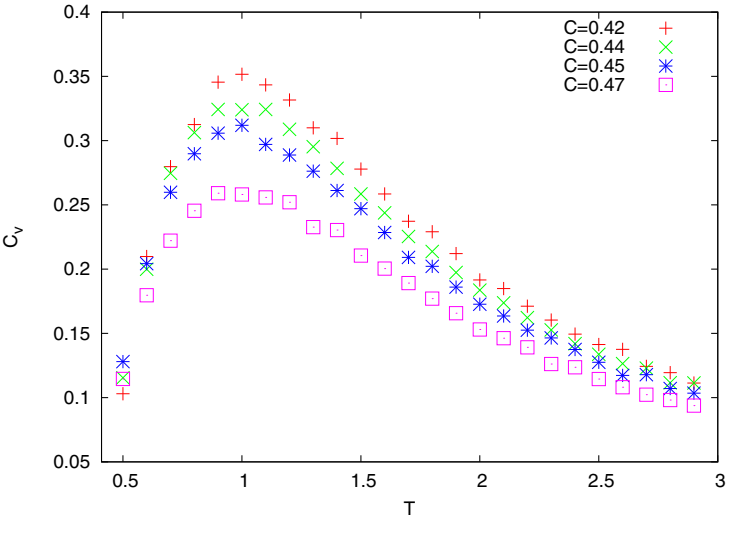


Fig. 3. The magnetic specific heat $C_v(T)$ for $< k >= 4$ and different values of the clustering coefficient C

specific heat $C_v(T)$ for the same network. The plots of the same quantities for $< k >= 9$ are shown in Figs. 4 and 5. The positions of the maxima of $\chi(T)$ and $C_v(T)$ can be treated as approximate values of the transition temperature T_{SG} [16,3]. Most curves displayed show some maxima except two cases with highest

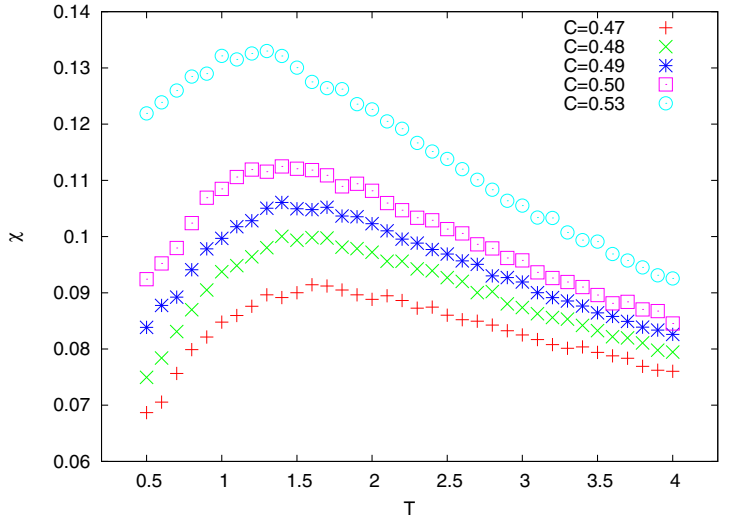


Fig. 4. The magnetic susceptibility $\chi(T)$ for $< k >= 9$ and different values of the clustering coefficient C

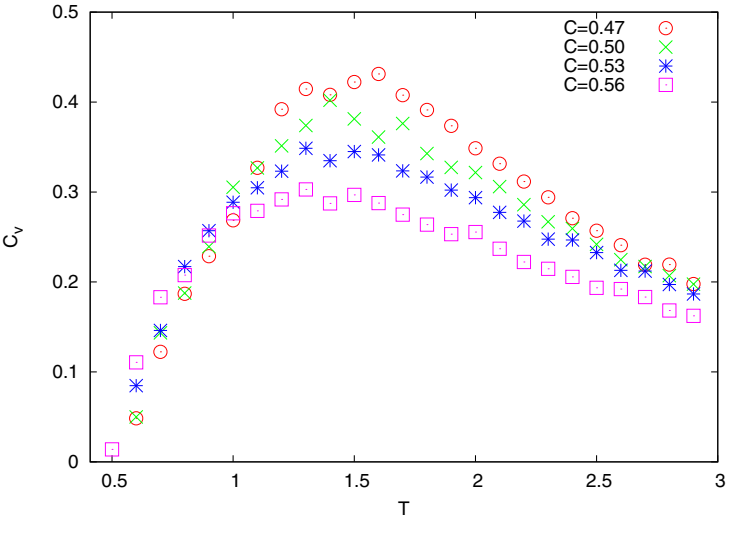


Fig. 5. The magnetic specific heat $C_v(T)$ for $< k >= 9$ and different values of the clustering coefficient C

C for $< k >= 4$, where the susceptibility for low temperatures does not decrease - this is shown in Fig. 2. Still it is clear that the observed maxima do not appear above $T = 1.1$ for $< k = 4 >$ and above $T = 1.7$ for $< k >= 9$. Moreover, the data suggest that when the clustering coefficient C increases, the positions of the maxima of $\chi(T)$ and $C_v(T)$ decrease. This is visible in particular for $< k >= 9$, in Figs. 4 and 5.

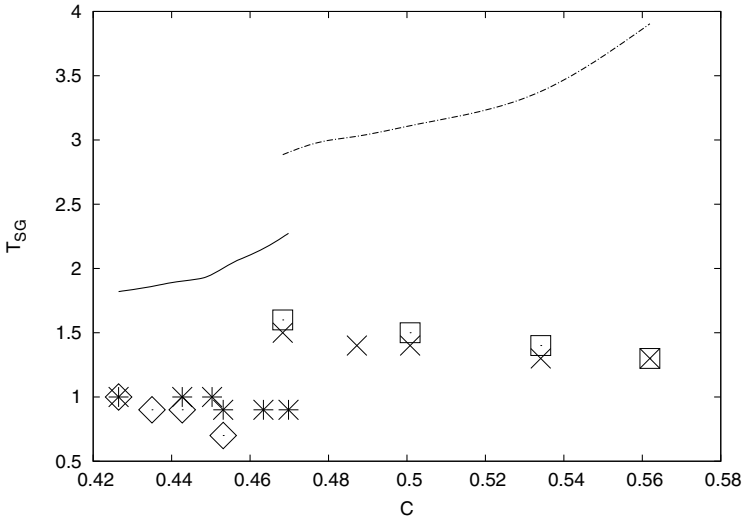


Fig. 6. The transition temperature T_{SG} for $< k >= 4$ (left side of the figure, continuous line) and $< k >= 9$ (right side, dotted line), against the clustering coefficient C. Points mark the results of the numerical simulations: for $< k >= 4$ stars come from χ and rhombs from C_v, and for $< k >= 9$ X's come from χ and squares from C_v. Lines are the theoretical plots from Eq. 1.

In Fig. 6 we show approximate values of the transition temperatures T_{SG}, as read from Figs. 2-5. These results are compared with the theoretical values of T_{SG}, obtained from Eq. 1. On the contrary to the numerical results, the Bethe theory indicates that T_{SG} is almost constant or increases with C. This discrepancy is our main numerical result. It is also of interest that once the clustering coefficient C increases, the susceptibility χ increases but the specific heat C_v decreases. This can be due with the variation of the shape of the free energy, as dependent on temperature and magnetic field.

4 Discussion

Our numerical results can be summarized as follows. The temperature T_{SG} of the transition from the paramagnetic phase to the spin-glass phase decreases with the clustering coefficient C. We interpret this decrease as a consequence of the

increase of the density of frustrations. More frustrated triads make the energy minima more shallow and then a smaller thermal noise is sufficient to throw the system from one to another minimum. This result is in contradiction to the Bethe theory. However, in this theory the frustration effect is neglected. Then the overall picture, obtained previously [10,11] for the small-world networks, is confirmed here also for the spatial networks.

This interpretation can be confronted with recent numerical results of Herrero, where the transition temperature T_{SG} increases with the clustering coefficient C in the square lattice [17]. As it is shown in Fig. 7 of [17], T_{SG} decreases from 2.3 to 1.7 when the rewiring probability p increases from zero to 0.4. Above $p = 0.4$, T_{SG} remains constant or increases very slightly, from 1.7 to 1.72 when $p = 1.0$. The overall dependence can be seen as a clear decrease of T_{SG}. On the other hand, the clustering coefficient C does not increase remarkably with the rewiring probability p. The solution of this puzzle is that in the square lattice with rewiring the frustrations are not due to triads, but to two interpenetrating sublattices, which are antiferromagnetically ordered in the case when $p = 0$. The conclusion is that it is the increase of the density of frustrations what always leads to a decrease of T_{SG}.

A few words can be added on the significance of these results for the science of complexity, with a reference to the computational problem of satisfiability. In many complex systems we deal with a number of external conditions, when all of them cannot be fulfilled. Second premise is that in many complex systems a noise is ubiquitous. These are analogs of frustration and thermal noise. In the presence of noise and contradictive conditions, the system drives in its own way between temporally stable states, similarly to the way how the Ising spin glass wanders between local minima of energy. Once the number of contradictive tendencies or aspirations increases, the overall structure becomes less stable.

Acknowledgements. We are grateful to Carlos P. Herrero for his comment. The calculations were performed in the ACK Cyfronet, Cracow, grants No. MNiSW /SGI3700 /AGH /030/ 2007 and MNiSW /SGI3700 /AGH /031/ 2007.

References

1. Dorogovtsev, S.N., Goltsev, A.V., Mendes, J.F.F.: Critical phenomena in complex networks. Rev. Mod. Phys. 80, 1275–1335 (2008)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability. In: A Guide to the Theory of NP-Completeness. W. H. Freeman and Comp., New York (1979)
3. Binder, K., Young, A.P.: Spin glasses: Experimental facts, theoretical concepts, and open questions. Rev. Mod. Phys. 58, 801–986 (1986)
4. Fischer, K.H., Hertz, J.A.: Spin Glasses. Cambridge UP, Cambridge (1991)
5. Newman, C.M., Stein, D.L.: Ordering and broken symmetry in short-ranged spin glasses. J. Phys.: Cond. Mat. 15, R1319–R1364 (2003)
6. Heermann, D.W.: Computer Simulation Methods in Theoretical Physics. Springer, Berlin (1986)

7. Krawczyk, M.J., Malarz, K., Kawecka-Magiera, B., Maksymowicz, A.Z., Kułakowski, K.: Spin-glass properties of an Ising antiferromagnet on the Archimedean $(3, 12^2)$ lattice. Phys. Rev. B 72, 24445 (2005)
8. Baxter, R.J.: Exactly Solved Models in Statistical Mechanics. Academic Press, London (1982)
9. Stauffer, D., Kułakowski, K.: Why everything gets slower? TASK Quarterly 7, 257–262 (2003)
10. Mańka, A., Malarz, K., Kułakowski, K.: Clusterization, frustration and collectivity in random networks. Int. J. Mod. Phys. C 18, 1765–1773 (2007)
11. Mańka-Krasoń, A., Kułakowski, K.: Magnetism of frustrated regular networks. Acta Phys. Pol. B (in Print) (arXiv:0812.1128)
12. Stanley, H.E.: Introduction to Phase Transitions and Critical Phenomena. Clarendon Press, Oxford (1971)
13. Aleksiejuk, A., Hołyst, J.A., Stauffer, D.: Ferromagnetic phase transitions in Barabási-Albert networks. Physica A 310, 260–266 (2002)
14. Herrmann, C., Barthélemy, M., Provero, M.: Connectivity distribution of spatial networks. Phys. Rev. E 68, 026128 (2003)
15. Holme, P., Kim, B.J.: Growing scale-free networks with tunable clustering. Phys. Rev. E 65, 026107 (2002)
16. Morgenstern, J., Binder, K.: Magnetic correlations in three-dimensional Ising spin glasses. Z. Physik B 39, 227–232 (1980)
17. Herrero, C.P.: Antiferromagnetic Ising model in small-world networks. Phys. Rev. E 77, 041102 (2008)

Weakness Analysis of a Key Stream Generator Based on Cellular Automata

Frédéric Pinel and Pascal Bouvry

University of Luxembourg
Faculty of Sciences, Communication and Technology
6, rue Coudenhove Kalergi
L-1359 Luxembourg-Kirchberg, Luxembourg
frederic.pinel@uni.lu, pascal.bouvry@uni.lu

Abstract. This paper exposes weaknesses of a secret-key cipher based on pseudo-random number generation. The pseudo-random number generator was previously described as high quality and passing various statistical tests (entropy, Marsaglia tests). It is operated by one-dimensional, two-state, non-uniform cellular automata with rules of radius one. Special rule assignments generate number sequences with zero entropy. The paper proposes a systematic construction that leads to such assignments, as well as the computation of the size of the weak key space. Finally, we envision solutions to this problem, and discuss the possibility to discover additional issues.

1 Introduction

Two types of cryptographic systems are used today: secret-key and public-key systems. An extensive overview of currently known or emerging cryptography techniques used in both types of systems can be found in [1]. One promising cryptographic technique is the use of cellular automata (CA). CA for secret-key systems were first studied by Wolfram [2], and later by Habutsu et al. [3], Nandi et al. [4] and Gutowitz [5]. More recently, they were a subject of study by Tomassini & Perrenoud [6], and Tomassini & Sipper [7], who considered one (1D) and two dimensional (2D) CA for their encryption scheme.

In this paper, we limit our study to secret-key systems. In such systems the encryption key and the decryption key are identical (and must therefore be kept secret). The secret-key encryption scheme we study is based on the generation of pseudo-random bit sequences. The bit sequence serves as the key stream. CA can effectively be used to generate pseudo-random number sequences. The 1D, non-uniform CA presented in [6] shows good statistical security characteristics. It indeed passes the classical Marsaglia tests [8]. Yet, the present article exposes a risk in the proposed system and shows potential paths for mitigating it. Such a risk comes from the existence in the key space of weak keys which lead to zero entropy bit sequences.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 547–552, 2010.

2 Cellular Automata and Cryptography

Secret-key cryptography uses the same key for encryption and decryption. A secret-key encryption scheme called the Vernam cipher is known to be [9,1] perfectly safe if the key stream is truly unpredictable and used only once. Let P be a plain-text message consisting of m bits $p_1p_2...p_m$, and $k_1k_2...k_m$ be a bit stream of a key k. Let c_i be the $i-th$ bit of a cipher-text obtained by applying XOR (exclusive-or) enciphering operation:

$$c_i = p_i \; XOR \; k_i.$$

The original plain-text bit p_i of a message can be recovered by applying a XOR operation on c_i with the same bit stream key k_i. As mentioned earlier, CA can produce pseudo-random number sequences which can serve as key streams.

We now provide definitions for CA in general and additional information specific to the CA-based random number generator presented in [6]. 1D CA consists of a lattice of cells. The values for each cell is restricted to a small, finite set of integers. Here, this set is $\{0, 1\}$. Each row of the CA corresponds to the cell space at a certain time step, and is called a *configuration*. The first row of the CA is the *initial configuration*. The value of each cell at a given time step is a function of the values of the neighboring cells, at the previous time step. This function is called a *rule*. The rule function is defined by specifying the "rule table" of values for every possible neighborhood. A neighborhood of radius r for cell i consists of $2r + 1$ cells: the cell i, which is updated by the function, and the r-cells adjacent to it in both directions. So, the value q_i^{t+1} of a cell i at time $t + 1$ is given by

$$q_i^{t+1} = f(q_{i-r}^t, q_{i-r+1}^t, \ldots, q_i^t, \ldots, q_{i+r-1}^t, q_{i+r}^t).$$

The *temporal sequence* [10] of a cell is defined as the successive values taken by the cell over time, $T_i = \{q_i^t, t = 0, 1, \ldots\}$. Wolfram proposed to name rules according to the decimal representation of the truth table. In Fig. 1, the truth table for a rule of radius 1, where values are in $\{0, 1\}$, can be represented as 01001011, this is the binary representation of the decimal number 75. In CA with a finite configuration size, a *cyclic boundary condition* results in a circular grid, where the cells located at the edges of the configuration are adjacent to each other. If all cells apply the same rule, the CA is said to be uniform. In contrast with a non-uniform CA, where different cells may apply different rules. The random number sequences in a CA are, for example, the different temporal

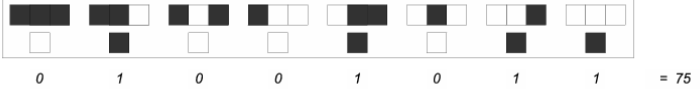


Fig. 1. Rule 75

sequences. The successive bits can be grouped together, producing a sequence of h-bit numbers, if h is the size of the group.

The CA-based random number generator presented in [6] is 1D, cyclic boundary, non-uniform with a neighborhood of radius $r = 1$ and with values for cells in $\{0, 1\}$. The rules cells can apply are 90, 105, 150, 165 (under Wolfram's rule naming convention described above). In [6], only the temporal sequence of the central cell is used, but it is common to allow the choice of the cell from which to read the sequence. In this paper, the cell from which to read the temporal sequence is not fixed and can be chosen freely.

3 Analysis of a Non-uniform CA for Random Number Generation

3.1 X-Weak Keys

In the system presented [6], a key is a choice of an initial configuration, a cell-to-rule mapping for every cell which defines the rule to apply at this cell, and the cell from which the random sequence will be read (its temporal sequence).

A weak key for a cryptosystem facilitates its cryptanalysis. In the context of our pseudo-random number generator, a weak key is an initial configuration and a cell-to-rule mapping such that the temporal sequence of the central cell displays repetition. This brings sequences of generated numbers to the complexity size $O(log(L))$, L being the original solution space. The set of rules is $\{90, 105, 150, 165\}$. The key space is then $L = N \times 2^N \times 4^N = N \times 2^{3N}$.

In this paper, we consider a stronger definition for weak keys. Let *x-weak* keys, for extremely weak keys, be this subset of the weak keys. An x-weak key is a choice of an initial configuration and a cell-to-rule mapping which leaves the initial configuration unchanged. Figure 2 illustrates this behavior.

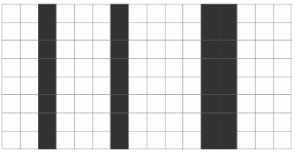


Fig. 2. Example of an x-weak key effect

Temporal sequences for all cells are a repetition of the same bit, the initial configuration's cell value. The entropy for such a sequence is 0. Since all cells produce a temporal sequence which is a repetition of one bit, an x-weak key does not need to include the chosen cell of the CA, as any key does.

3.2 Production of X-Weak Keys

In spite of the excellent random number generation properties previously reported in [6], such x-weak keys exist. One trivial x-weak key is a configuration

Table 1. Truth tables

Neighborhood	90	105	150	165
000	0	1	0	1
001	1	0	1	0
010	0	0	1	1
011	1	1	0	0
100	1	0	1	0
101	0	1	0	1
110	1	1	0	0
111	0	0	1	1

where all cells are 0, and apply rule 90. Besides this trivial key, which may not be encountered given its regularity (although it is a valid key for the CA scheme considered), there are other x-weak keys.

Table 1 presents the truth tables of the rules cells can apply. It can be observed that for every possible neighborhood, there are always 2 rules out of 4, which leave the cell unchanged. Let an *identity* rule be a rule which leaves the cell unchanged, for a given neighborhood. Since an x-weak key must leave the entire configuration unchanged, all cells must be assigned an identity rule.

A simple procedure to construct an x-weak key is to first choose a random initial configuration. Then for each cell, determine its neighborhood and map it to one of the 2 identity rules matching this neighborhood. Figure 3 illustrates this construction.

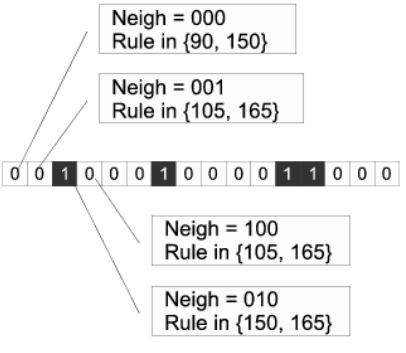


Fig. 3. Constructing an x-weak key

3.3 Size of X-Weak Key Space

When selecting a key under this CA scheme, the probability to choose an identity rule is $\frac{1}{2}$, for any neighborhood. Therefore for any configuration, each cell can be mapped to an identity rule with probability $\frac{1}{2}$. If the 1D CA is of size N, then the probability to choose an x-weak key is $\frac{1}{2^N}$. As the random number sequence

is read from any chosen cell, and not from the central cell, the x-weak key space K is:

$$K = L \times \frac{1}{2^N} = N \times 2^{3N} \times \frac{1}{2^N} = N \times 2^{2N}.$$

3.4 Other Weak Keys

In addition to x-weak keys defined above, other types of weak keys exist. A *transient* x-weak key produces good random sequences for each cell until a given configuration is reached, after which the rules applied keep repeating this configuration. The entropy for the temporal sequences is zero from this step onwards. Table 1 shows that any of the 4 rules is an identity rule for half of the neighborhoods. So at any time step, the probability to leave a cell unchanged exists, as well as the probability to leave the configuration unchanged.

Also, even if only a part of configuration is changed, the more cells are left unchanged, the lower the entropy of the temporal sequences becomes.

3.5 Risk Mitigation

There are several ways to mitigate the risk of x-weak keys.

The x-weak key probability is $\frac{1}{2^N}$, so increasing the value for N reduces the probability. With a value for N of 100 or greater, this probability is sufficiently low.

If the value for N cannot be increased then another way to mitigate the risk is to reduce the probability of mapping a cell to an identity rule. This probability directly depends on the rule set of the non-uniform CA. The rule set is obtained through a cellular programming approach [11], where CA rules with the desired properties (random temporal sequences in this case) are evolved. Each cellular programming run evolves rules (through their truth table) to discover those which produce good random temporal sequences. The rules discovered most often compose the rule set. They are not evaluated collectively. Therefore this process cannot be adapted to mitigate the x-weak key risk, which results from the collective behavior of the rule set. In [12], an evolutionary algorithm is used to identify the subset of rules which shows the best collective behavior, from a given set of rules previously discovered by cellular programming. The fitness function of this algorithm can be modified to mitigate the x-weak key risk. An indication of the probability to map a cell to an identity rule can be weighted in the original fitness function, which computes the average entropy of temporal sequences of all cells, over several initial configurations. This indication is, for example, the number of identity rules over the total rule set, averaged over all neighborhoods.

4 Conclusion

In this paper, we extended the results reported in [6] by analyzing the weak keys of a secret-key cryptographic system based on a non-uniform, one-dimension,

cellular automata. The CA is used to generate random number sequences. We defined extremely weak keys (x-weak), quantified their presence in the key space, and proposed counter measures based on [12]. If the probability to encounter such x-weak keys is low, their effect is the generation of a key stream with zero entropy. Future work will include the analysis of other non-uniform CA-based random number generators.

References

1. Schneier, B.: Applied Cryptography. Wiley, Chichester (1996)
2. Wolfram, S.: Cryptography with cellular automata. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 429–432. Springer, Heidelberg (1986)
3. Habutsu, T., Nishio, Y., Sasase, I., Mori, S.: A secret key cryptosystem by iterating a chaotic map. In: Davies, D.W. (ed.) EUROCRYPT 1991. LNCS, vol. 547, pp. 127–140. Springer, Heidelberg (1991)
4. Nandi, S., Kar, B.K., Chaudhuri, P.P.: Theory and applications of cellular automata in cryptography. IEEE Trans. Computers 43(12), 1346–1357 (1994)
5. Gutowitz, H.: Cryptography with dynamical systems. In: Goles, E., Boccara, N. (eds.) Cellular Automata and Cooperative Phenomena. Kluwer Academic Press, Dordrecht (1993)
6. Tomassini, M., Perrenoud, M.: Stream cyphers with one- and two-dimensional cellular automata. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) PPSN 2000. LNCS, vol. 1917, pp. 722–731. Springer, Heidelberg (2000)
7. Tomassini, M., Sipper, M., Perrenoud, M.: On the generation of high-quality random numbers by two-dimensional cellular automata. IEEE Trans. Computers 49(10), 1146–1151 (2000)
8. Marsaglia, G.: Diehard (1998), http://www.stat.fsu.edu/pub/diehard/
9. Menezes, A., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, Boca Raton (1996)
10. Jen, E.: Aperiodicity in one-dimensional cellular automata. In: Gutowitz, H. (ed.) Cellular Automata, Theory and Experiment, vol. 45, pp. 3–18. Physica D/MIT (1990)
11. Sipper, M.: Evolution of Parallel Cellular Machines. In: Sipper, M. (ed.) Evolution of Parallel Cellular Machines. LNCS, vol. 1194. Springer, Heidelberg (1997)
12. Szaban, M., Seredynski, F., Bouvry, P.: Collective behavior of rules for cellular automata-based stream ciphers. In: IEEE Congress in Evolutionary Computation (2006)

Fuzzy Cellular Model for On-Line Traffic Simulation

Bartłomiej Płaczek

Faculty of Transport, Silesian University of Technology,
ul. Krasinskiego 8, 40-019 Katowice, Poland
bartlomiej.placzek@polsl.pl

Abstract. This paper introduces a fuzzy cellular model of road traffic that was intended for on-line applications in traffic control. The presented model uses fuzzy sets theory to deal with uncertainty of both input data and simulation results. Vehicles are modelled individually, thus various classes of them can be taken into consideration. In the proposed approach, all parameters of vehicles are described by means of fuzzy numbers. The model was implemented in a simulation of vehicles queue discharge process. Changes of the queue length were analysed in this experiment and compared to the results of NaSch cellular automata model.

1 Introduction

Road traffic models are often used in on-line mode to determine current and actual traffic parameters as well as to forecast future state of the flow for traffic control purposes. The term on-line means that the traffic model works in real time and gathers current traffic data acquired by vehicle detectors. An example can be given here of an on-line traffic model that uses counts of detected vehicles to calculate discharge time of a vehicles queue in a crossroad approach. The anticipation ability of traffic parameters is essential for actuated and synchronized traffic control.

Available systems of adaptive traffic control (e.g. SCOOT [1], UTOPIA [2]) use traffic models that describe queues or groups of vehicles rather than individual cars and their parameters. However, individual characteristics related to class of a vehicle are very important from the traffic control point of view, as they have a significant influence on traffic conditions and capacity of road infrastructure.

Modern traffic control systems are mostly intended for cooperation with traffic detectors that recognise presence or passing of vehicles and count them (e.g. inductive loops). Additional functionalities offered by vision based sensors cannot be fully utilised in systems of this kind [3]. Video-detection technology is usually simply adopted as a substitute for inductive loops, thus important data available for vision based sensors is discarded.

Properties of particular vehicles are considered when using microscopic traffic models. Computationally efficient and sufficiently accurate models have been

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 553–560, 2010.

developed based on the cellular automata theory [4]. In the literature there is a lack of information about experiments on cellular models implementation in traffic control systems.

In this paper a fuzzy cellular traffic model is introduced. This model was intended for on-line applications in traffic control. It enables utilisation of complex traffic data registered by vision based sensors. Development of this model was motivated by the following requirements.: (1) A vehicle influence on traffic conditions depends on its individual features, thus various classes of vehicles have to be modelled. (2) The traffic model has to provide data interfaces for many sources - detectors of different types. (3) The uncertainty has to be described in traffic model to take into account random nature of traffic processes as well as rough character of vehicles recognition (detection) results. (4) Computational complexity of the model has to be appropriately low to allow on-line processing.

The rest of this paper is organised as follows. Section 2 includes a brief survey of cellular automata applications in road traffic modelling. A fuzzy cellular traffic model is introduced in section 3. In section 4 simulation results of vehicles queue discharge process are discussed for the proposed model and compared with those of NaSch cellular automaton. Finally, in section 5 conclusions are drawn.

2 Cellular Automata in Traffic Modelling

Due to their simplicity, cellular automata have become a frequently used tool for microscopic modelling of road traffic processes. Traffic models using cellular automata have high computational efficiency and allow sufficiently accurate simulation of real traffic phenomena [5].

The cellular automaton model for road traffic simulation was introduced by Nagel and Schreckenberg (NaSch) [4] in 1992. In this model the traffic lane was divided into cells of 7,5 m. Each vehicle is characterised by its position (cell number) and velocity (a positive discrete value lower than fixed maximum). The velocity is expressed as a number of cells that vehicle advances in one time step. Movement of vehicles in this model is described by a simple rule that is executed in parallel for each vehicle. It has the capability of mapping of real traffic streams parameters (fundamental diagram) and enables simulation of phenomena that can be observed in reality (e.g. traffic jam formation).

In the literature many traffic models can be found that are based on the Nagel-Schreckenberg concept. Numerous models have been introduced that uses so-called slow-to-start rules to reflect metastable state of traffic flow [6,7,8,9]. According to the slow-to-start (s2s) rule stopped vehicles accelerate with lower probability than moving ones. Different rules of this kind takes into account various factors: number of free cells in front of a vehicle (gap), present or previous state of a vehicle (velocity). In [10] a new set of rules has been proposed to better capture driver reactions to traffic that are intended to preserve safety on the highway.

On the basis of the NaSch cellular automaton multi-lane traffic models have been formulated [11,12,13,14]. Rules of these models comprise two steps: first,

additional step takes into account lanes changing behaviour and second, basic step describes forward movement of a vehicle.

Cellular automata have also been used for junctions modelling. The simplest cellular model of a crossroad [15] did not take into account region inside the junction as well as priority rules. Vehicles were just randomly selected to pass the crossroad. In [16] other simple model has been proposed of cellular automaton with closed boundary conditions (ring of cells) for crossroads simulation. All junctions in this case were modelled as roundabouts. For more realistic simulations sophisticated models have been applied that include definitions of traffic regulations (priority rules, signs, signalisation) and allow for determination of actual junction capacity [17].

Little research work has been done in the application of microscopic cellular models for traffic control in road networks. Certain methods have been proposed of optimal route selection in urban networks [18]. The cellular automata model of road network has been used in this approach to evaluate current traffic conditions for particular connections. In [19] similar model was adopted to calculate basic parameters of coordination plan for signalised intersections network. However, the analysed cases were significantly simplified and far from practical solutions.

In the field of road traffic modelling several methods are known using cells defined as road segments for macroscopic flow description. Although the term cell is used in these methods, they are not derived from cellular automata theory. One cell in this case can be occupied by many vehicles thus its state is described using parameters of traffic stream (density, intensity). A model of this type was implemented for traffic control purposes in the UTOPIA method [2], different models have been introduced for highway traffic analysis [20].

3 Fuzzy Cellular Model of Road Traffic Flow

The fuzzy cellular model of road traffic flow is proposed as an extension of the NaSch traffic model. It assumes a division of traffic lane into cells that correspond to road segments of equal length. The traffic state is described in discrete time steps. Vehicle position, its velocity and other parameters are modelled as fuzzy numbers defined on the set of integers.

State of a cell c in time step t is defined by fuzzy set of vehicles (n) that currently occupy this cell:

$$S_{c,t} = \left\{ \mu_{S_{c,t}}(n)/n \right\}. \tag{1}$$

Thus, one cell can be occupied by more than one vehicle in the same time. Conventionally, $\mu_A(x)$ denotes value of membership function of fuzzy set A for an element x. Position of vehicle n in time step t is a fuzzy number defined on the set of cells indexes (c):

$$P_{n,t} = \left\{ \mu_{P_{n,t}}(c)/c \right\}. \tag{2}$$

Vehicle n is described by its class and velocity$V_{n,t}$ (in cells per time step). The class determines properties of vehicle n: length L_n (in cells), maximal velocity

V_n^{max}, and acceleration A_n. All these quantities are expressed by fuzzy numbers. Velocity of the vehicle n in time step t is computed according to formula:

$$V_{n,t} = \widetilde{\min}\left\{V_{n,t-1}\widetilde{+}A_n, G_{n,t}, V_n^{max}\right\}. \tag{3}$$

The tilde ($\widetilde{\ }$) symbol is used to distinguish operations on fuzzy numbers [21]. Gap $G_{n,t}$ is the number of free cells in front of vehicle n:

$$G_{n,t} = \widetilde{\min_{m \neq n}}\left\{P_{m,t}\widetilde{-}L_m\widetilde{-}P_{n,t} : P_{m,t}\widetilde{>}P_{n,t}\right\}. \tag{4}$$

If there is no vehicle m fulfilling the condition in (4), gap $G_{n,t}$ is assumed to be equal to the maximal velocity V_n^{max}.

Position of vehicle n in the next time step $(t+1)$ is computed on the basis of the model state in time t:

$$P_{n,t+1} = \widetilde{dil}\left(P_{n,t}\widetilde{+}V_{n,t}\right), \tag{5}$$

$\widetilde{dil}$ denotes fuzzy set dilation:

$$\mu_{\widetilde{dil}\left(P_{n,t}\widetilde{+}V_{n,t}\right)}(x) = \left[\mu_{\left(P_{n,t}\widetilde{+}V_{n,t}\right)}(x)\right]^e, \tag{6}$$

where $0 < e \leq 1$.

Dilating the fuzzy set increases the fuzziness (uncertainty) of the vehicles position. This operation corresponds to the randomization step of traffic models based on NaSch cellular automaton. In the models that use s2s rules the randomization level decreases with increasing velocity of a vehicle as the random driver behaviours are more intense at low velocity range. To achieve similar effect for the presented model the exponent e in (6) was defined as an increasing function of velocity. It was also assumed that when the maximal velocity is reached the vehicle position is no further dilated ($e = 1$). A simple linear dependency was used to control the dilation:

$$e = \alpha + \frac{1-\alpha}{\hat{v}_n^{max}}\hat{v}_{n,t}, \tag{7}$$

where $0 \leq \alpha \leq 1$ and $\hat{v}$ denotes defuzzified (crisp) value of velocity:

$$\hat{v}_{n,t} = \arg\max_y \mu_{V_{n,t}}(y). \tag{8}$$

Fig. 1 presents results of a traffic simulation that was performed using the fuzzy cellular model. Simulation was started with single vehicle ($n = 0$) stopped in the first cell ($c = 0$): $P_{0,0} = \{1/0\}$, $V_{0,0} = \{1/0\}$, the vehicle properties was set: $L_0 = \{1/0\}$, $V_0^{max} = \{0, 2/4; 1/5; 0, 2/6\}$, $A_0 = \{0, 2/0; 1/1; 0, 2/2\}$. Space-time diagrams in fig. 1 depict how the vehicle accelerates, its fuzzy positions are showed using gray levels. If the colour is darker for a cell, the value of membership function of fuzzy set $P_{0,t}$ is higher in this cell (white colour indicates empty cells, black indicates cells where $\mu_{P_{0,t}}(c) = 1$).

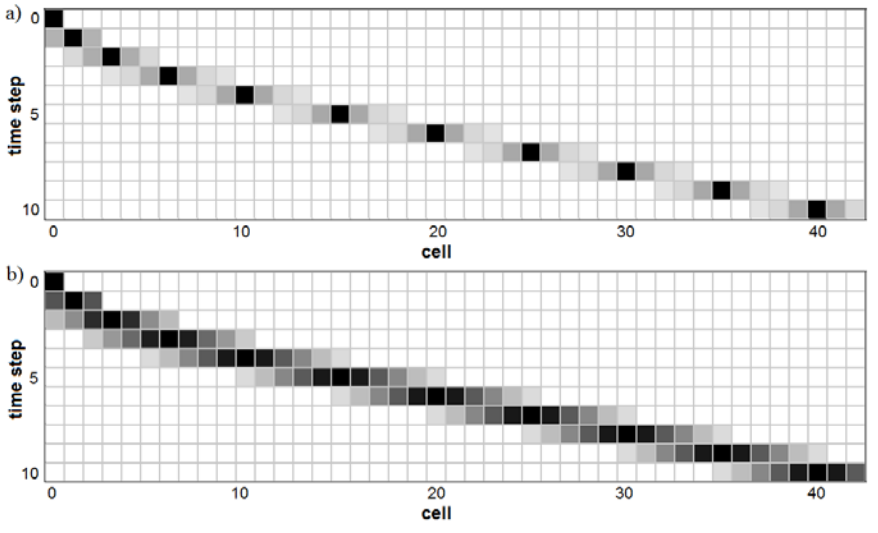


Fig. 1. Space-time diagram for the single-vehicle simulation: a) $\alpha = 0,9$ b) $\alpha = 0,1$

Results of the single vehicle movement simulation are compared in fig. 1 for two values of parameter α , which was used for controlling the dilation (eq. 7). It shows that decreasing value of α causes increase in fuzziness of the vehicle position (higher level of model uncertainty).

4 Vehicles Queue Modelling

The fuzzy cellular model presented in previous section was applied for a simulation of vehicles queue discharge process that corresponds to real-life situations observed at approaches of signalised crossroads when green signal is given. During this experiment length of vehicles queue was analysed in sequence of time steps. This section includes discussion of the simulation results as well as their comparison with experimental data obtained using NaSch traffic model.

Only one class of vehicles was used in this simulation defining their maximal velocity: $V_{max} = \{0, 2/2; 1/3; 0, 2/4\}$ and remaining properties that were set identically to those of single-vehicle experiment reported in section 2. Dilation operation (6) was applied with parameter $\alpha = 0,9$. In the first step of simulation all fifty vehicles ($n = 0 \ldots 49$) were stopped in a queue: $P_{n,0} = \{1/n\}$, $V_{n,0} = \{1/0\}$, it means that at the beginning length of the queue was equal to the number of vehicles.

In subsequent time steps the traffic model was updated according to equations (3), (5) and the queue length Q_t was evaluated. For the introduced traffic model Q_t is defined as fuzzy value using following fuzzy rule:

if veh_0 is in_queue and veh_1 is in_queue and...and $veh\_x - 1$ is in_queue and veh_x is not in_queue and...and veh_m is not in_queue then Q_t is x, (9)

where veh_n stands for "vehicle n", m denotes number of vehicles and variable in_queue is determined by another fuzzy rule taking into account position and velocity of a vehicle:

$$\text{if } P_{n,t} \text{ is } n \text{ and } V_{n,t} \text{ is } 0 \text{ then } veh\_n \text{ is } in\_queue. \tag{10}$$

Results of vehicles queue length computations based on fuzzy cellular simulation are presented in fig 2 a), gray scale was used in this case to depict membership function value of Q_t (darker colour correspond to higher value). These results are compared with experimental data on vehicles queue discharge that was collected from traffic simulation driven by NaSch cellular automaton (fig. 2 b).

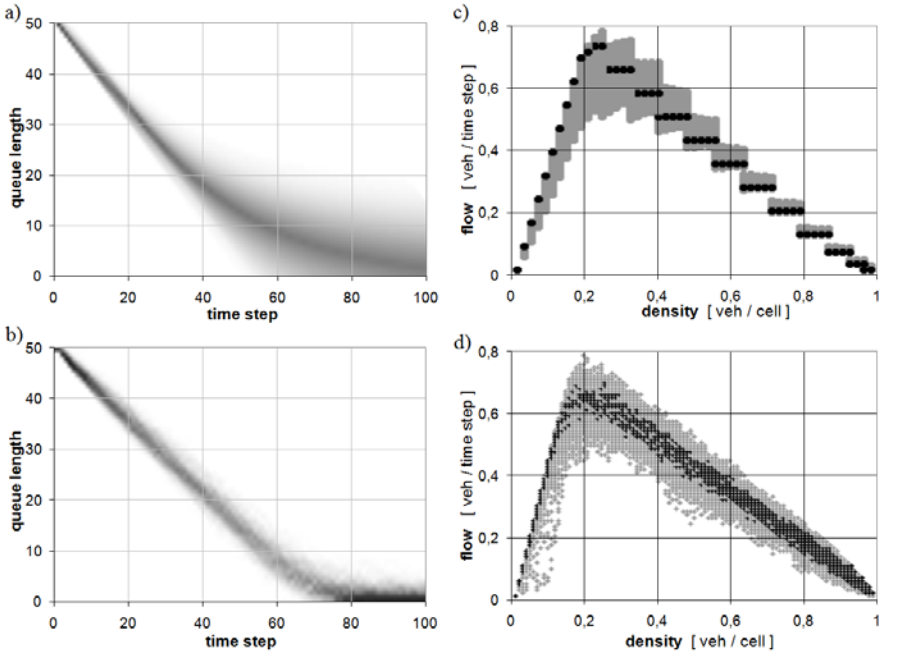


Fig. 2. Results of the vehicles queue discharge simulation

Vehicles characteristics as well as starting conditions for the simulation using NaSch model were similar to those defined for fuzzy cellular simulation. Probabilistic parameter p of the NaSch model was set to 0,2 and v_{max} was 3. Simulation was executed over two hundred times to gather the data presented in fig. 2 b). Gray levels in this chart correspond to experimental probability of specific queue length evaluated for a given time step of the simulation.

The fuzzy cellular model was validated with regard to the traffic flow theory by comparing fundamental diagrams (dependency between traffic density and flow volume). Experimental data for the NaSch model (fig. 4 d) was collected from 200 executions of traffic simulation. Black dots in this diagram indicate

traffic states that have experimental probability higher or equal 0,1, gray dots correspond with lower probabilities. Figure 4 c) includes diagram obtained for the fuzzy cellular model. Flow volume was determined as a fuzzy number for each value of traffic density. The black dots in this diagram indicate maximal values of the flow membership functions and the gray lines correspond to alfa-cuts computed using threshold value of 0,99.

Comparison of the simulation results (fig. 2) shows that proposed fuzzy cellular model adequately describes the process of traffic flow. It should be noted that single simulation using fuzzy cellular model gives comparable results to the distribution of experimental probability computed for many NaSch simulations. The proposed model inherently describes uncertainty of traffic states. Conventional cellular automaton needs many instances of the model and additional statistics to explore uncertainty of traffic parameters i.e. range of their possible values.

5 Conclusions

Fuzzy cellular model of road traffic was formulated on the basis of cellular automata approach. Parameters of vehicles are individually described by means of fuzzy numbers, thus various classes of vehicles can be modelled. Application of fuzzy sets theory allows to represent uncertainty of traffic states using single instance of the model. It takes into account random nature of traffic processes and makes the model suitable for collecting imprecise data from traffic detectors.

The uncertainty is described applying fuzzy definitions of vehicles parameters. E.g. uncertain position of a vehicle is expressed by a fuzzy number defined on the set of cells indexes. This uncertain position in the model can correspond with imprecise results of vehicle detection. Thus, the main advantage of fuzzy approach is the model's ability to utilise imprecise traffic data for on-line simulation.

Experimental results of the vehicles queue discharge simulation reported in this contribution show that fuzzy cellular model adequately describes the process of traffic flow. The queue lengths and fundamental diagrams were analysed in this experiment and compared to the results of NaSch cellular automata model. Further tests are necessary to evaluate the model applicability for other real-traffic situations. Planned research will also involve design of communication procedures that are needed to input data from traffic detectors into the model.

References

1. Martin, P.T., Hockaday, S.L.M.: SCOOT: An update. ITE Journal 65(1), 44–48 (1995)
2. Mauro, V., Taranto, C.: UTOPIA. In: Proceedings of the 6th IFAC/IFORS Conf. on Control, Computers and Communications in Transport, Paris, pp. 245–252 (1989)
3. Placzek, B., Staniek, M.: Model Based Vehicle Extraction and Tracking for Road Traffic Control. In: Kurzynski, M., et al. (eds.) Advances in Soft Computing. Computer Recognition Systems, vol. 2, pp. 844–851. Springer, Heidelberg (2007)

4. Nagel, K., Schreckenberg, M.: A cellular automaton model for freeway traffic. J. Physique I 2, 2221–2241 (1992)
5. Placzek, B.: The method of data entering into cellular traffic model for on-line simulation. In: Piecha, J. (ed.) Trans. on Transport Systems Telematics, pp. 34–41. Publishing House of Slesian Univ. of Technology, Gliwice (2006)
6. Barlovic, R., Santen, L., Schadschneider, A., Schreckenberg, M.: Metastable states in cellular automata for traffic flow. The European Physical Journal B 5(3), 793–800 (1998)
7. Chowdhury, D., Santen, L., Schadschneider, A.: Statistical physics of vehicular traffic and some related systems. Physic Reports 329, 199–329 (2000)
8. Emmerich, H., Rank, E.: An improved cellular automaton model for traffic flow simulation. Physica A 234(3-4), 676–686 (1997)
9. Pottmeier, A., Berlovic, R., Knopse, W., Schadschneider, A., Schreckenberg, M.: Localized defects in a cellular automaton model for traffic flow with phase separation. Physica A 308(1-4), 471–482 (2002)
10. Shih-Ching, L., Chia-Hung, H.: Cellular Automata Simulation for Traffic Flow with Advanced Control Vehicles. In: The 11th IEEE Int. Conf. on Computational Science and Engineering Workshops, pp. 328–333. IEEE, Los Alamitos (2008)
11. Rickert, M., Nagel, K., Schreckenberg, M., Latour, A.: Two Lane Traffic Simulations using Cellular Automata. Physica A 231(4), 534–550 (1995)
12. Knopse, W., Santen, L., Schadschneider, A., Schreckenberg, M.: Disorder in cellular automata for two-lane traffic. Physica A 265(3-4), 614–633 (1999)
13. Wagner, P., Nagel, K., Wolf, D.E.: Realistic Multi-Lane Traffic Rules for Cellular Automata. Physica A 234(3-4), 687–698 (1996)
14. Xianchuang, S., Xiaogang, J., Yong, M., Bo, P.: Study on Asymmetric Two-Lane Traffic Model Based on Cellular Automata. In: Sunderam, V.S., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2005. LNCS, vol. 3514, pp. 599–606. Springer, Heidelberg (2005)
15. Rickert, M., Nagel, K.: Experiences with a Simplified Microsimulation for the Dallas/Fort Worth Area. Int. J. of Modern Physics C 8(3), 483–503 (1997)
16. Dupuis, A., Chopard, B.: Parallel simulation of traffic in Geneva using cellular automata. In: Kuhn, E. (ed.) Virtual shared memory for distributed architectures, pp. 89–107. Nova Science Publishers, New York (2001)
17. Esser, J., Schreckenberg, M.: Microscopic simulation of urban traffic based on cellular automata. Int. J. of Modern Physics C 8(5), 1025–1036 (1997)
18. Wahle, J., Annen, O., Schuster, C., Neubert, L., Schreckenberg, M.: A dynamic route guidance system based on real traffic data. European Journal of Operational Research 131(2), 302–308 (2001)
19. Brockfeld, E., Barlovic, R., Schadschneider, A., Schreckenberg, M.: Optimizing traffic lights in a cellular automaton model for city traffic. Physical Review E 64(056132), 1–12 (2001)
20. Daganzo, C.: The cell transmission model. Part II: Network traffic. Transportation Research B 29(2), 79–93 (1995)
21. Dubois, D., Prade, H.: Operations on fuzzy numbers. International Journal of Systems Science 9(6), 613–626 (1978)

Modeling Stop-and-Go Waves in Pedestrian Dynamics

Andrea Portz and Armin Seyfried

Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany

Abstract. Several spatially continuous pedestrian dynamics models have been validated against empirical data. We try to reproduce the experimental fundamental diagram (velocity versus density) with simulations. In addition to this quantitative criterion, we tried to reproduce stop-and-go waves as a qualitative criterion. Stop-and-go waves are a characteristic phenomenon for the single file movement. Only one of three investigated models satisfies both criteria.

1 Introduction

The applications of pedestrians' dynamics range from the safety of large events to the planning of towns with a view to pedestrian comfort. Because of the computational effort involved with an experimental analysis of the complex collective system of pedestrians' behavior, computer simulations are run. Models continuous in space are one possibility to describe this complex collective system.

In developing a model, we prefer to start with the simplest case: single lane movement. If the model is able to reproduce reality quantitatively and qualitatively for that simple case, it is a good candidate for adaption to complex two-dimensional conditions.

Also in single file movement pedestrians interact in many ways and not all factors, which have an effect on their behavior, are known. Therefore, we follow three different modeling approaches in this work. All of them underlie diverse concepts in the simulation of human behavior.

This study is a continuation and enlargement of the validation introduced in [7]. For validation, we introduce two criteria: On the one hand, the relation between velocity and density has to be described correctly. This requirement is fulfilled, if the modeled data reproduce the fundamental diagram. On the other hand, we are aiming to reproduce the appearance of collective effects. A characteristic effect for the single file movement are stop-and-go waves as they are observed in experiments [5]. We obtained all empirical data from several experiments of the single file movement. There a corridor with circular guiding was built, so that it possessed periodic boundary conditions. The density was varied by increasing the number of the involved pedestrians. For more information about the experimental set-up, see [5],[6].

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 561–568, 2010.

2 Spatially Continuous Models

The first two models investigated are force based and the dynamics are given by the following system of coupled differential equations

$$m_i \frac{d\,v_i}{d\,t} = F_i \quad \text{with} \quad F_i = F_i^{drv} + F_i^{rep} \quad \text{and} \quad \frac{d\,x_i}{d\,t} = v_i \tag{1}$$

where F_i is the force acting on pedestrian i. The mass is denoted by m_i, the velocity by v_i and the current position by x_i. This approach derives from sociology [4]. Here psychological forces define the movement of humans through their living space. This approach is transferred to pedestrians dynamics and so F_i is split to a repulsive force F_i^{rep} and a driven force F_i^{drv}. In our case the driving force is defined as

$$F_i^{drv} = \frac{v_i^0 - v_i}{\tau}, \tag{2}$$

where v_i^0 is the desired velocity of a pedestrian and τ their reaction time.

The other model is event driven. A pedestrian can be in different states. A change between these states is called event. The calculation of the velocity of each pedestrian is straightforward and depends on these states.

2.1 Social Force Model

The first spatially continuous model was developed by Helbing and Molnár [3] and has often been modified. According to [2] the repulsive force for pedestrian i is defined by

$$F_i^{rep}(t) = \sum_{j \neq i} f_{ij}(x_i, x_j) + \xi_i(t) \quad \text{with} \quad f_{ij}(x_i, x_j) = -\partial_x A(\Delta\, x_{i,j} - D)^{-B}, \tag{3}$$

with $A = 0.2, B = 2, D = 1\,[m], \tau = 0.2\,[s]$ and $\Delta\, x_{i,j}$ is the distance between pedestrians i and j. The fluctuation parameter $\xi_i(t)$ represents the noise of the system. In two-dimensional scenes, this parameter is used to create jammed states and lane formations [2]. In this study, we are predominantly interested in the modeled relation between velocity and density for single file movement. Therefore the fluctuation parameter has no influence and is ignored.

First tests of this model indicated that the forces are too strong, leading to unrealistically high velocities. Due to this it is necessary to limit the velocity to v_{max}, as it is done in [3]

$$v_i(t) = \begin{cases} v_i(t), & \text{if } |v_i(t)| \leq v_{max} \\ v_{max}, & \text{otherwise} \end{cases} . \tag{4}$$

In our simulation we set $v_{max} = 1.34\,[\frac{m}{s}]$.

2.2 Model with Foresight

In this model pedestrians possess a degree of foresight, in addition to the current state of a pedestrian at time step t. This approach considers an extrapolation of the velocity to time step $t+s$. For it [8] employs the linear relation between the velocity and the distance of a pedestrian i to the one in front $\Delta x_{i,i+1}(t)$.

$$v_i(t) = a\,\Delta x_{i,i+1}(t) - b \tag{5}$$

For $a = 0.94\,[\frac{m}{s}]$ and $b = 0.34\,[\frac{1}{s}]$ this reproduces the empirical data. So with (5) $v_i(t+s)$ can be calculated from $\Delta x_{i,i+1}(t+s)$ which itself is a result of the extrapolation of the current state

$$\Delta x_{i,i+1}(t) + \Delta v(t)\,s = \Delta x_{i,i+1}(t+s) \tag{6}$$

with $\Delta v(t) = v_{i+1}(t)\,s - v_i(t)\,s$. Finally, the repulsive force is defined as

$$F_i^{rep}(t) = -\frac{v_i^0 - v_i(t+s)}{\tau}\ . \tag{7}$$

Obviously the impact of the desired velocity v_i^0 in the driven force is negated by the one in the repulsive term. After some simulation time, the system reaches an equilibrium in which all pedestrians walk with the same velocity. In order to spread the values and keep the right relation between velocity and density, we added a fluctuation parameter $\zeta_i(t)$. $\zeta_i(t)$ uniformly distributed in the interval $[-20, 20]$ reproduced the scatter observed in the empirical data.

2.3 Adaptive Velocity Model

In this model pedestrians are treated as hard bodies with a diameter d_i [1]. The diameter depends linearly on the current velocity and is equal to the step length st in addition to the safety distance β

$$d_i(t) = e + f\,v_i(t) = st_i(t) + \beta_i(t)\ . \tag{8}$$

Based on [9] the step length is a linear function of the current velocity with following parameters:

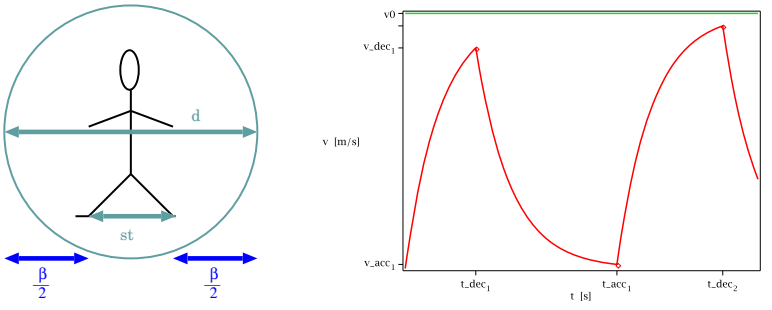
$$st_i(t) = 0.235\,[m] + 0.302\,[s]\,v_i(t)\ . \tag{9}$$

e and f can be specified through empirical data and the inverse relation of (5). Here e is the required space for a stationary pedestrian and f affects the velocity term. For $e = 0.36\,[m]$ and $f = 1.06\,[s]$ the last equations (8) and (9) can be summarized to

$$\beta_i(t) = d_i(t) - st_i(t) = 0.125\,[m] + 0.758\,[s]\,v_i(t)\ . \tag{10}$$

By solving the differential equation

$$\frac{dv}{dt} = F^{drv} = \frac{v^0 - v(t)}{\tau} \quad \Rightarrow \quad v(t) = v^0 + c\,\exp\left(-\frac{t}{\tau}\right), \text{for } c \in \mathbb{R}, \tag{11}$$



(a) Demonstration of the parameters d, st and β

(b) Conception of the adaptive velocity

Fig. 1. Left: connection between the required space d, the step length st and the safety distance β. Right: The adaptive velocity with acceleration until t_{dec1}, than deceleration until t_{acc1}, again acceleration until t_{dec2} and so on.

the velocity function is obtained. This is shown in Fig. 1 together with the parameters of the pedestrians' movement.

A pedestrian is accelerating to their desired velocity v_i^0 until the distance to the pedestrian in front is smaller than the safety distance. From this time on, he/she is decelerating until the distance is larger than the safety distance and so on. Via $\Delta x_{i,i+1}, d_i$ and β_i those events could be defined: deceleration (12) and acceleration (13).

To ensure good performance for high densities, no events are explicitly calculated. But in each time step, it is checked whether an event has taken place and t_{dec}, t_{acc} or t_{coll} are set to t accordingly. The time step, Δt, of 0.05 seconds is chosen, so that a reaction time is automatically included. The discrete time step could lead to configurations where overlapping occurs. To guarantee volume exclusion, case (14) is included, in which the pedestrians are too close to each other and have to stop.

$$t = t_{dec}, \quad \text{if:} \quad \Delta x_{i,i+1} - 0.5 * (d_i(t) + d_{i+1}(t)) \leq 0 \tag{12}$$

$$t = t_{acc}, \quad \text{if:} \quad \Delta x_{i,i+1} - 0.5 * (d_i(t) + d_{i+1}(t)) > 0 \tag{13}$$

$$t = t_{coll}, \quad \text{if:} \quad \Delta x_{i,i+1} - 0.5 * (d_i(t) + d_{i+1}(t)) \leq -\beta_i(t) \tag{14}$$

3 Validation with Empirical Data

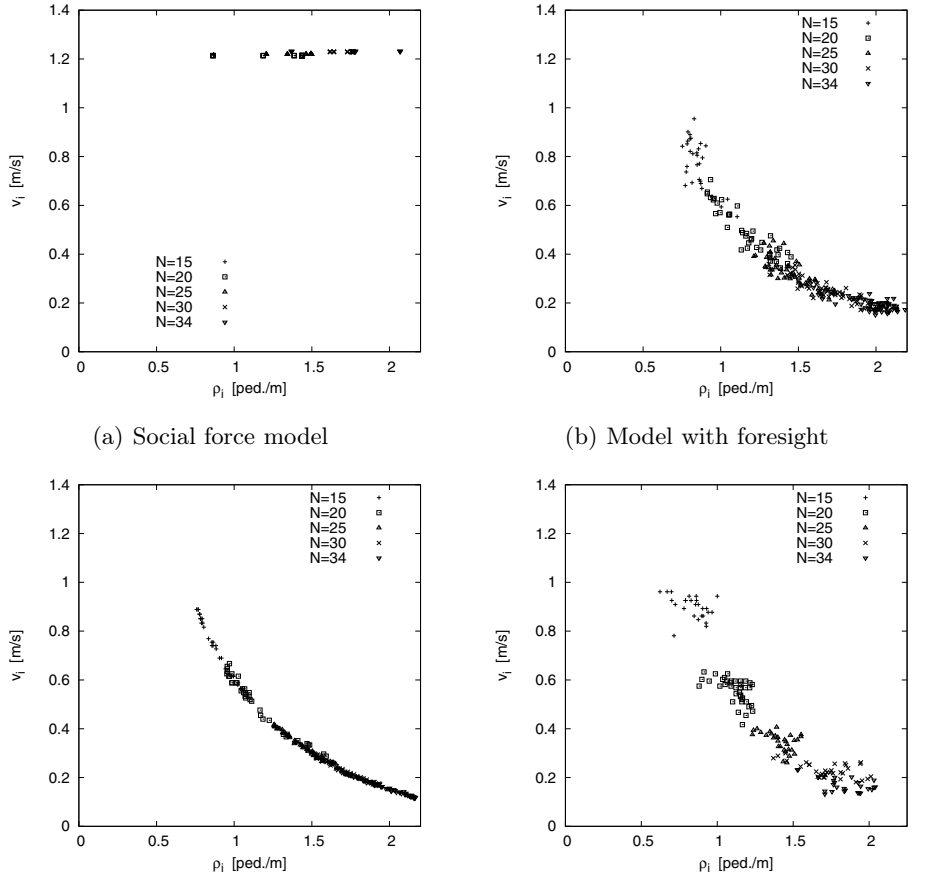
For the comparison of the modeled and experimental data, it is important to use the same method of measurement. [5] shows that the results from different measurement methods vary considerably. The velocity v_i is calculated by the entrance and exit times t_i^{in} and t_i^{out} to two meter section.

$$v_i = \frac{2\,[m]}{(t_i^{out} - t_i^{in})[s]}. \tag{15}$$

To avoid discrete values of the density leading to a large scatter, we define the density by

$$\rho(t) = \frac{\sum_{i=1}^{N} \Theta_i(t)}{2\,m}, \tag{16}$$

where $\Theta_i(t)$ gives the fraction to which the space between pedestrian i and $i+1$ is inside the measured section, see [6]. ρ_i is the mean value of all $\rho(t)$, where t is in the interval $[t_i^{in}, t_i^{out}]$. We use the same method of measurement for the modeled and empirical data. The fundamental diagrams are displayed in Fig. 2, where N is the number of the pedestrians.



(a) Social force model

(b) Model with foresight

(c) Adaptive velocity model

(d) Empirical data

Fig. 2. Validation of the modeled fundamental diagram with the empirical data (down right) for the single file movement

The velocities of the social force model are independent of the systems density and nearly equal to the desired velocity $v_i = v_i^0 \sim 1.24\,[\frac{m}{s}]$. Additionally we observe a backward movement of the pedestrians and pair formation. Because of these unrealistic phenomena are not observed in the other models, we suggest that this is caused by the combination of long-range forces and periodic boundary conditions.

In contrast, the model with foresight results in a fundamental diagram in good agreement with the empirical one. Through the fluctuation parameter, the values of the velocities and densities vary as in the experimental data.

We are satisfied with the results of the adaptive velocity model. For reducing computing time, we also tested a linear adaptive velocity function, leading to a 70% decrease in computing time for 10000 pedestrians. The fundamental diagram for this linear adaptive velocity function is not shown, but also reproduces the empirical one.

4 Reproduction of Stop-and-Go Waves

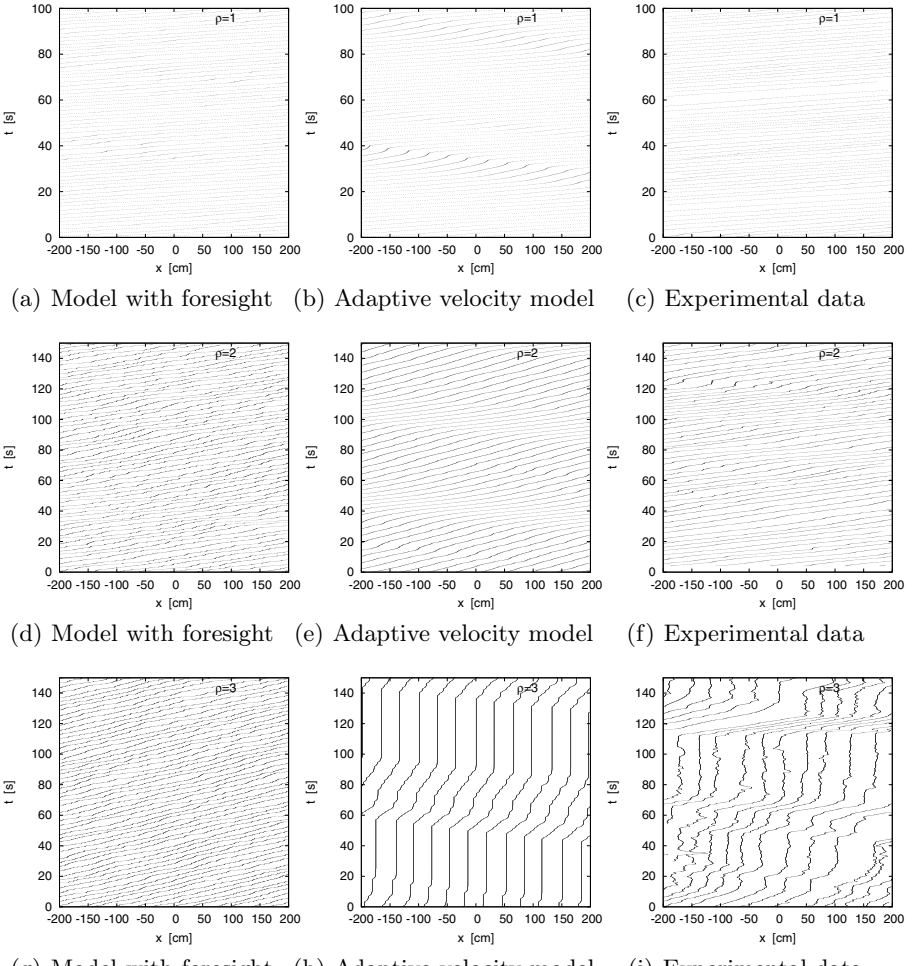
During the experiments of the single file movement, we observed stop-and-go waves at densities higher than two pedestrians per meter, see Fig. 5 in [5]. Therefore, we compare the experimental trajectories with the modeled ones for global densities of one, two and three persons per meter. The results are shown in Fig.3. Since the social force model is not able to satisfy the criterion for the right relation between velocity and density, we omit this model in this section. Figure 3 shows the trajectories for global average densities of one, two and three persons per meter. From left to right the data of the model with foresight, the adaptive velocity model and the experiment are shown.

In the experimental data, it is clearly visible that the trajectories get unsteadier with increasing density. At a density of one person per meter pedestrians stop for the first time. So a jam is generated. At a density of two persons per meter stop-and-go waves pass through the whole measurement range. At densities greater than three persons per meter pedestrians can hardly move forward.

For the extraction of the empirical trajectories, the pedestrians' heads were marked and tracked. Sometimes, there is a backward movement in the empirical trajectories caused by self-dynamic of the pedestrians' heads. This dynamic is not modeled and so the other trajectories have no backward movement. This has to be accounted for in the comparison.

By adding the fluctuation parameter $\zeta_i(t) \in [-20, 20]$ to the model with foresight a good agreement with experiment is obtained for densities of one and two persons per meter. The irregularities caused by this parameter are equal to the irregularities of the pedestrians dynamic. Nevertheless, this does not suffice for stopping so that stop-and-go waves appear, Fig.3(d) and Fig.3(g).

With the adaptive velocity model stop-and-go waves already arise at a density of one pedestrian per meter, something that is not seen in experimental data. However, this model characterizes higher densities well. So in comparison with Fig. 3(h) and Fig. 3(i) the stopping-phase of the modeled data seems to last for



(a) Model with foresight (b) Adaptive velocity model (c) Experimental data

(d) Model with foresight (e) Adaptive velocity model (f) Experimental data

(g) Model with foresight (h) Adaptive velocity model (i) Experimental data

Fig. 3. Comparison of modeled and empirical trajectories for the single lane movement. The global density of the system is one, two or three persons per meter (from top to bottom).

the same time as in the empirical data. But there are clearly differences in the acceleration phase, the adaptive velocity models acceleration is much lower than seen in experiment.

Finally other studies of stop-and-go waves have to be carried out. The occurrence of this phenomena has to be clearly understood for further model modifications. Therefore it is necessarry to measure e. g. the size of the stop-and-go wave at a fixed position. Unfortunately it is not possible to measure over a time interval, because the empirical trajectories are only available in a specific range of 4 meters.

5 Conclusion

The well-known and often used social force model is unable to reproduce the fundamental diagram. The model with foresight provides a good quantitative reproduction of the fundamental diagram. However, it has to be modified further, so that stop-and-go waves could be generated as well. The model with adaptive velocities follows a simple and effective event driven approach. With the included reaction time, it is possible to create stop-and-go waves without unrealistic phenomena, like overlapping or interpenetrating pedestrians.

All models are implemented in C and run on a simple PC. They were also tested for their computing time in case of large system with upto 10000 pedestrians. The social force model offers a complexity level of $\mathcal{O}(N^2)$, whereas the other models only have a level of $\mathcal{O}(N)$. For this reason the social force model is not qualified for modeling such large systems. Both other models are able to do this, where the maximal computing time is one sixth of the simulated time.

In the future, we plan to include steering of pedestrians. For these models more criteria, like the reproduction of flow characteristics at bottlenecks, are necessarry. Further we are trying to get a deeper insight into to occurrence of stop-and-go waves.

References

1. Chraibi, M., Seyfried, A.: Pedestrian Dynamics With Event-driven Simulation. In: Pedestrian and Evacuation Dynamics (2008/2009), arXiv:0806.4288 (in Print)
2. Helbing, D., Farkas, I.J., Vicsek, T.: Freezing by Heating in a Driven Mesoscopic System. Phys. Rev. Let. 84, 1240–1243 (2000)
3. Helbing, D., Molnár, P.: Social force model for pedestrian dynamics. Phys. Rev. E 51, 4282–4286 (1995)
4. Lewin, K. (ed.): Field Theory in Social Science. Greenwood Press, New York (1951)
5. Seyfried, A., Boltes, M., Kähler, J., Klingsch, W., Portz, A., Schadschneider, A., Steffen, B., Winkens, A.: Enhanced empirical data for the fundamental diagram and the flow through bottlenecks. In: Pedestrian and Evacuation Dynamics 2008. Springer, Heidelberg (2008/2009) (in print), arXiv:0810.1945
6. Seyfried, A., Steffen, B., Klingsch, W., Boltes, M.: The fundamental diagram of pedestrian movement revisited. J. Stat. Mech., P10002 (2005)
7. Seyfried, A., Steffen, B., Lippert, T.: Basics of modelling the pedestrian flow. Physica A 368, 232–238 (2006)
8. Steffen, B., Seyfried, A.: The repulsive force in continous space models of pedestrian movement (2008), arXiv:0803.1319v1
9. Weidmann, U.: Transporttechnik der Fussgänger. Technical Report Schriftenreihe des IVT Nr. 90, Institut für Verkehrsplanung,Transporttechnik, Strassen- und Eisenbahnbau, ETH Zürich, ETH Zürich, Zweite, ergänzte Auflage (1993)

FPGA Realization of a Cellular Automata Based Epidemic Processor

Pavlos Progias, Emmanouela Vardaki, and Georgios Ch. Sirakoulis

Laboratory of Electronics, Department of Electrical and Computer Engineering,
Democritus University of Thrace, 67100 Xanthi, Greece
(pp8646,ev4925,gsirak)@ee.duth.gr

Abstract. More and more attention is paid to epidemics of infectious diseases that spread through populations across large regions and under condition may result on increased mortality in the infected population. In this paper, a FPGA processor based on Cellular Automata (CA) able to produce successive epidemic fronts, as it is expected by theory, is presented. The proposed CA SIR (Susceptible, Infectious and Recovered) model successfully includes the effect of movement of individuals, on the epidemic propagation as well as the effect of population vaccination which reduces the epidemic spreading. The FPGA design results from the automatically produced synthesizable VHDL code of the CA model and is advantageous in terms of low-cost, high speed and portability. Finally, the resulted hardware implementation of the CA model is considered as basic component of a wearable electronic system able to provide real time information concerning the epidemic propagation on the under test part of the examined population.

Keywords: Epidemic Spreading, Cellular Automata, SIR, FPGA.

1 Introduction

In epidemiology, an epidemic (from Greek words epi- upon + demos people) occurs when new cases of a certain disease spread in a given human population, during a given period, substantially exceed what is "expected", based on recent experience [1]. An epidemic may be restricted to one locale (an outbreak), more general (an "epidemic") or even, more critical, global (pandemic). According to the World Health Organization (WHO), a pandemic can start when three conditions meet [2]: Emergence of a disease new to a population, agents infect humans, causing serious illness and agents spread easily and sustainably among humans. Famous examples of epidemics have been recorded in mankind history and include HIV (present), the pandemic of the 14th century known as the Black Death, the Great Influenza Pandemic which coincided with the end of World War I and many mores. However, even in recent days, with the 2009 outbreak of a new strain of Influenza A virus sub-type H1N1, the WHO's pandemic alert level, which has been three for some years, was moved to four and till this moment at five. On the other hand, several of the existing computational intelligence

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 569–574, 2010.

techniques can be adopted for solving epidemic problems, and new methods are developed. In general, epidemics have been modeled using differential equations [3]. However this approach has some serious drawbacks [5]: it neglects the local character of the spreading process, it does not include variable susceptibility of individuals, and can not handle the complicated boundary and initial conditions. Cellular Automata (CAs) [4] are an alternative to partial differential equations and they can easily handle complicated boundary and initial conditions, inhomogeneities and anisotropies [5]. Furthermore, CAs can overcome the above drawbacks, and consequently have been used by several researchers as an alternative method of modeling epidemics [5,6].

In this paper, a CA model able to simulate epidemic spreading in which the state of the cell is obtained from the fraction of the number of individuals which are susceptible, infected, or recovered from the disease is presented [5,6]. Moreover, the proposed two-dimensional (2-d) CA SIR (Susceptible, Infectious and Recovered) model successfully includes the effect of movement of individuals, in a hypothetical homogeneous population, on the epidemic propagation as well as the effect of population vaccination which reduces the epidemic spreading. The distance of movement and the number of individuals that are going to move are the two most important parameters, which are taken into consideration by our model. The increment of the aforementioned parameters resulted in our CA in the same way, as it is theoretical assigned, i.e. when the percentage of the moving population is increased, or the maximum distance of population movement is increased, the epidemic fronts lose their symmetry (i.e. their circular shape) and the spreading of the epidemic disease is accelerated. Furthermore, because of the inherent parallelism of CAs, the proposed model is hardware implemented with the help of Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) synthesizable code in order to speed up the application of CA to the study of epidemic spreading. More specifically, a translation algorithm is used, that checks the CA parameters values previously determined by the user and automatically produces the synthesizable VHDL code that describes the aforementioned CA. It should be mentioned that CAs are one of the computational structures best suited for hardware realization. In this paper, the design processing of the finally produced VHDL code, i.e. analysis, elaboration and simulation, has been checked out with the help of the Quartus II design software of the ALTERA Corporation. Consequently, the implementation of the resulting VHDL code results in a FPGA, which is considered as basic component of a wearable electronic system able to provide real time information concerning the epidemic propagation on the under test part of the examined population. More specifically, taking into account GPS (Global Position System) tracking, the resulted processor is fed with real data about possible roads and railways routes for population movement and feedbacks with a support decision system providing on time information regarding the possible epidemic propagation. The proposed wearable electronic system should be also equipped with wi-fi transceiver-transmitter for communication reasons as well as with proper detectors corresponding to the certain epidemic characteristics.

2 Modeling of Epidemic Spreading in Terms of CA

As mentioned before, differential equation models were used to describe epidemic models from the early beginnings of previous century, e.g. the Kermack and McKendrick (1927) SIR model [3], with the system of ordinary differential equations.

$$\dot{S} = -aSI,\ \dot{I} = aSI - bI,\ \dot{R} = bI \tag{1}$$

where S is the susceptible part, I the infected part, and R the recovered part of the population. Here it is assumed that the population is well mixed, an assumption which in reality is not valid. Also, it is assumed that the total population is constant, i.e. external effects, such as death, movement, imported objects etc., are neglected. The variable susceptibility of individuals is also neglected. In order to avoid such the CA approach is used. More specifically, the population over which the epidemic propagation will be modeled is assumed to exist in a 2-d space and it is homogenous. The 2-d space is divided into a matrix of identical square cells, with side length a, and it is represented by a CA [5]. Each CA cell includes a number of individuals living there. The number of spatial dimensions of the CA array is: $n = 2$. The widths of the two sides of the CA array are taken to be equal, i.e. $w_1 = w_2$. The size of the array (i.e. the values of $w_{1,2}$) is defined by the user of the model, and it is a compromise between accuracy and computation time. The width of the neighborhood of a CA cell is taken to be equal to 3 in both array sides, i.e. $d_1 = d_2 = 3$.

The state $C^t_{i,j}$ of the (i, j) CA cell, at time t, is:

$$C^t_{i,j} = \left\{P^t_{i,j}, INF^t_{i,j}, IMF^t_{i,j}\right\} \tag{2}$$

where $INF^t_{i,j}$ is a flag called the infectious flag. The value of this flag indicates whether some of the individuals located in the (i, j) cell are infected by the disease at time t. If $INF^t_{i,j} = 1$, then $P^t_{i,j}$ is the fraction of the number of individuals in the (i, j) cell infected by the disease, at time t:

$$P^t_{i,j} = \frac{S^t_{i,j}}{T^t_{i,j}} \tag{3}$$

where $S^t_{i,j}$ is the infected part of the population and $T^t_{i,j}$ is the total population in the (i, j) cell. $P^t_{i,j}$ may take any value between 0 and 1, but, in order to keep the number of states of the CA finite, $P^t_{i,j}$ is made to take 21 discrete values leveling from 0.00 to 0.05 and so on till 1.00. Obviously, if $INF^t_{i,j} = 0$, then $P^t_{i,j} = 0$.

The time duration of the disease is user defined and it can be assumed to be equal to t_{in}. After that time the population has recovered from the disease and has acquired a temporal immunity to this disease. After t_{in} time steps, the flag $INF^t_{i,j}$ will change its value from 1 to 0. At this time the flag $IMF^t_{i,j}$ will also change its value from 0 to 1. $IMF^t_{i,j}$ is called the immune flag and indicates whether the population located in the (i, j) cell is immune to the disease or not. The immune population loses its immunity after time t_{im}, it becomes again

susceptible to the disease, and the value of the flag $IMF^t_{i,j}$ becomes equal to 0. The time duration t_{im} is also user defined. Recapitulating, the CA cells may be found in one of the following three general states:

1. If $INF^t_{i,j} = 0$, and $IMF^t_{i,j} = 0$, then the population in the (i, j) cell is susceptible to the disease.
2. If $INF^t_{i,j} = 1$, $IMF^t_{i,j} = 0$, and $0 < P^t_{i,j} \leq 1$, then the population in the (i, j) cell is infected.
3. If $INF^t_{i,j} = 0$, and $IMF^t_{i,j} = 1$, then the population in the (i, j) cell is immune to the disease.

Each CA cell starts as susceptible and becomes infected, if some CA cell in its neighborhood is infected. It remains infected for time t_{in}, and then it becomes immune. It remains immune for time t_{im}, and then it becomes susceptible again. The transition from the susceptible state to the infected state is done according to:

$$P^{t+1}_{i,j} = P^t_{i,j} + k\left(P^t_{i-1,j}, P^t_{i,j-1}, P^t_{i,j+1}, P^t_{i+1,j}\right) + l\left(P^t_{i-1,j-1}, P^t_{i-1,j+1}, P^t_{i+1,j-1}, P^t_{i+1,j+1}\right) \tag{4}$$

In Eq. 4, the effect of the adjacent nearest neighbors is multiplied by k, whereas the effect of the diagonal adjacent neighbors is multiplied by l. It is expected that the (i, j) cell will be infected more quickly, if it has an infected adjacent nearest neighbor, than if it has an infected diagonal adjacent neighbor, because of the more extensive contact between populations. Therefore, it is always $k > l$ [5].

As mentioned before, the distance of movement and the number of individuals that are going to move are the two most important parameters in case of population movement. In all examing cases the central CA cell was assumed to be infectious and that it spreads the epidemic to its neighborhood, only once. Different percentages of the population that was about to move and different maximum distances of movement *max_distance* were taken into account. The possible directions of population movement could be given in accordance to the road and railway connection of the geographical area under study. Finally, in order to simulate the effect of vaccination, we assumed that a small part of the initial population is vaccinated. In all simulations the CA model was able to produce successive epidemic fronts, as it was expected by theory.

3 FPGA Implementation

After the performance and the functional correctness of the CA epidemic model is checked, a CA translation algorithm, written in a high-level scripting language, is used. This translation algorithm receives the programming code of the CA model originally written in Matlab as its input, and automatically produces, as output, a synthesizable VHDL code. The final VHDL code produced by translation algorithm, including both the behavioral and structural parts, addresses the basic VHDL concepts. To achieve its goal, the translation algorithm collects information from the disease CA model by checking its primary

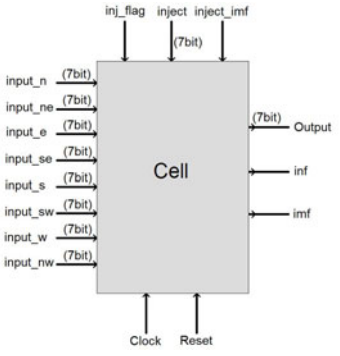


Fig. 1. The block diagram of the CA cell with its inputs and outputs

parameters. To be more specific, the entity declaration of a CA cell describes the input/output ports of the module, which happens to be the main component in our VHDL code. As shown in Fig. 1, the contributions of the North, Northeast, ..., and Northwest neighbours to the change of the state of the CA cell are loaded through *input_s*, *input_n*, *input_ne*, ..., *input_nw*, while *inject_imf* stands for the value of immunity flag placed into the cell, *inj_flag* enables loading of values into the cell, and *inject* input loads an initial value into the cell. In correspondence, the state of the cell after the CA rule execution at the next time step is given by *output*, while output signals inf and imf stand for the infected flag and immunity flag, respectively. The architecture body of the behavioral part of VHDL code displays the implementation of the entity CA cell. Subsequently, the translation algorithm searches the CA code to detect the lattice size, the boundary and initials CA conditions, in order to construct the structural part of the final VHDL code. The structural part implements the final module as a composition of subsystems, like the aforementioned main component. In addition, it includes component instances of previously declared entity/architecture pairs, port maps in components and wait statements.

The automatically produced synthesizable VHDL CA code is translated into a hardware schematic of the defined architecture using predetermined timing constraints in Quartus II, v. 7.2 design software. Design of the proposed processor results in an ALTERA Stratix EP1S60F1020C5 FPGA device, which indicates a maximum clock rate around 100MHz, consists of 250 CA cells and uses 73% of the total available logic elements. Initial data can be loaded in a semi-parallel way and the automatic response of the processor provides the CA epidemic propagation. Furthermore, there is always a possibility of functional simulation of the VHDL code with the use of the appropriate automatically generated test benches as presented at Fig. 2. As a result, the simulation results of the VHDL code are found in complete agreement with the compilation results of the CA model.

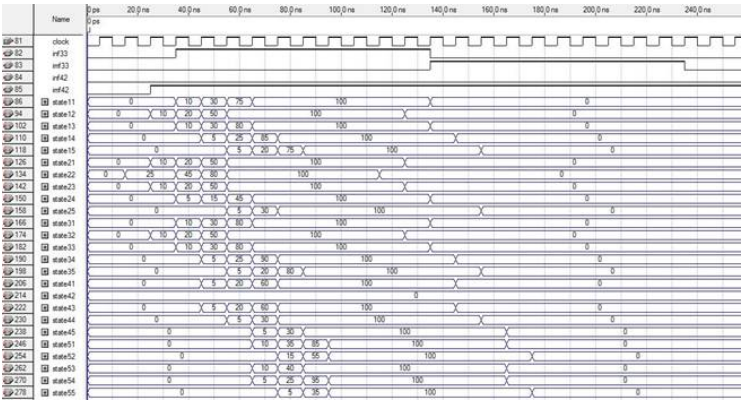


Fig. 2. Timing diagram of the presented FPGA processor for modelling epidemic spreading

4 Conclusions

In this paper, a FPGA processor able to reproduce the effects of population movement and vaccination on epidemic propagation is presented. The FPGA design results from the automatically produced synthesizable VHDL code of the CA model and succeeds to maximize the CA performance, brings reliability and low total cost to its potential user and enables portability by its low power dissipation. The later is crucial since, the resulted FPGA is considered as basic component of a wearable electronic system able to provide real time information concerning the epidemic propagation on the under test part of the examined population. As a result, the proposed FPGA could serve as the basis of a support decision system for monitoring epidemic propagation under condition depending on the rest characteristics of the electronic system.

References

1. Definition of Epidemic in Wikipedia, http://en.wikipedia.org/Epidemic
2. World Health Organization: Changing History. The World Health Report 2004, Geneva (2004)
3. Kermack, W.O., McKendrick, A.G.: A contribution to the mathematical theory of epidemics. Proc. R. Soc. A115, 700–721 (1927)
4. Wolfram, S.: Cellular Automata and Complexity. Addison-Wesley, Reading (1994)
5. Sirakoulis, G.C., Karafyllidis, I., Thanailakis, A.: A cellular automaton model for the effect of population movement on epidemic propagation. Ecological Modelling 133(3), 209–223 (2000)
6. del Rey, Á.M., Hoya White, S., Sánchez, G.R.: A model based on cellular automata to simulate epidemic diseases. In: El Yacoubi, S., Chopard, B., Bandini, S. (eds.) ACRI 2006. LNCS, vol. 4173, pp. 304–310. Springer, Heidelberg (2006)

Empirical Results for Pedestrian Dynamics at Bottlenecks

Armin Seyfried[1] and Andreas Schadschneider[2]

[1] Jülich Supercomputing Centre, Forschungszentrum Jülich, 52425 Jülich, Germany
a.seyfried@fz-juelich.de
[2] Institut für Theoretische Physik, Universität zu Köln, 50937 Köln, Germany
as@thp.uni-koeln.de

Abstract. In recent years, several approaches for modeling pedestrian dynamics have been developed. Usually the focus is on the qualitative reproduction of empirically observed collective phenomena like the dynamical formation of lanes. Although this gives an indication of the realism of a model, for practical applications as in safety analysis reliable quantitative predictions are required. This asks for reliable empirical data. Here we discuss the current status of one of the basic scenarios, the dynamics at bottlenecks. Here there is currently no consensus even about the qualitative features of bottleneck flows.

keywords: empirical data, bottlenecks, fundamental diagram.

1 Introduction

The investigation of pedestrian dynamics is not only of scientific interest, e.g. due to the various collective phenomena that can be observed, but also of great practical relevance [1]. Therefore it is quite surprising that the empirical and experimental situation is still unsatisfactory. Even for the most basic quantities used for the characterization of pedestrian streams no consensus has been reached. For the flow-density relation, usually called fundamental diagram, properties like the maximal possible flow (capacity) or the density of complete flow breakdown vary by factors of 2 to 4 in different studies and even in safety guidelines [2,3,4,5,6,7,8,9].

The situation for one of the most important scenario in pedestrian dynamics, the behaviour at bottlenecks where the maximal flow or throughput is locally reduced, is very similar. A thorough understanding of such situation is highly relevant for evacuations where typically the bottlenecks have a strong influence on the evacuation time. Here even the qualitative dependence of the capacity on the bottleneck width is disputed. Some guidelines and studies find a step-wise increase whereas others find strong evidence for a continuous increase.

Obviously these issues have to be settled to increase the reliability of any safety analysis. On the other hand, a consensus on the empirical results is essential for the validation and calibration of modelling approaches. So far most models have

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 575–584, 2010.

only been tested qualitatively, e.g. by their ability to reproduce the observed collective phenomena like lane formation in counterflow situations.

In the following we will discuss various aspects of measurements on pedestrian streams. After introducing the basic quantities and discussing the influence of the measurement methods we will focus on bottleneck experiments. We will report results from recent large-scale laboratory experiments that have been performed under controlled conditions [10,11,12].

2 Basic Quantities of Pedestrian Flows

The main characteristic quantities for the description of pedestrian streams are flow and density. The flow J of a pedestrian stream is defined as number of pedestrians crossing a fixed location of a facility per unit of time. It can be measured in different ways. The most natural approach determines the times t_i at which pedestrians have passed a fixed measurement location. The flow is then calculate from the time gaps $\Delta t_i = t_{i+1} - t_i$ between two consecutive pedestrians i and $i+1$:

$$J = \frac{1}{\langle \Delta t_i \rangle} \qquad \text{with} \qquad \langle \Delta t_i \rangle = \frac{1}{N} \sum_{i=1}^{N} (t_{i+1} - t_i)\,. \tag{1}$$

The analogy with fluid dynamics suggest another method to measure the flow of a pedestrian stream. It relates the flow through a facility of width b with the average density ρ and the average speed v of the pedestrian stream,

$$J = \rho\, v\, b = J_s\, b\,, \tag{2}$$

where the *specific flow*[1]

$$J_s = \rho\, v \tag{3}$$

gives the flow per unit-width. This relation is also known as *hydrodynamic relation*. The quantities involved can be obtained by determining entrance and exit times for a test section, e.g. from video recordings or time-lapse photography. This allows to calculate the velocity of each pedestrian. The associated density $\rho = N/A$ is measured by counting the number of pedestrians N within the selected area A at the time when the moving pedestrian was at the center of the section.

Another way to quantify the pedestrian load of facilities has been proposed by Fruin [13]. The "pedestrian area module" is given by the reciprocal of the density. Predtechenskii and Milinskii [2] consider the ratio of the sum of the projection area f_j of the bodies and the total area of the pedestrian stream A, defining the (dimensionless) density

$$\tilde{\rho} = \frac{\sum_j f_j}{A}\,. \tag{4}$$

[1] In strictly one-dimensional motion often a line density (dimension: 1/length) is used. Then the *flow* is given by $J = \rho v$.

Since the projection area f_j depends strongly on the type of person (e.g. it is much smaller for a child than an adult), the densities for different pedestrian streams consisting of the same number of persons and the same stream area can be quite different. An alternative definition is introduced in [9] where the local density is obtained by averaging over a circular region of radius R, $\rho(\boldsymbol{r},t) = \sum_j f(\boldsymbol{r}_j(t) - \boldsymbol{r})$, where $\boldsymbol{r}_j(t)$ are the positions of the pedestrians j in the surrounding of $\boldsymbol{r}$ and $f(...)$ is a Gaussian, distance-dependent weight function.

Most of these measurements combine an *average* velocity or flow with some *instantaneous* density. This is an additional factor why measurements in similar settings can differ in a large way. Beside technical problems due to camera distortions and camera perspective there are several conceptual problems, like the association of averaged with instantaneous quantities, the necessity to choose an observation area in the same order of magnitude as the extent of a pedestrian together with the definition of the density of objects with non zero extent and much more. A detailed analysis how the way of measurement could influences the relations is outlined in the next section.

3 Influence of the Measurement Method

To demonstrate the magnitude of the variations due to different measurement methods we choose the simplest possible system, namely the movement of pedestrians along a line with closed boundary conditions. Examples for exemplary trajectories can be found elsewher in these procedings, see [14]. The situation is similar to that of vehicular traffic which allows us to adopt the discussion in [15,16] to the case of pedestrian streams. In the following we will discuss the two principle approaches to measure observables like flow, velocity and density. These are illustrated in in Fig. 1.

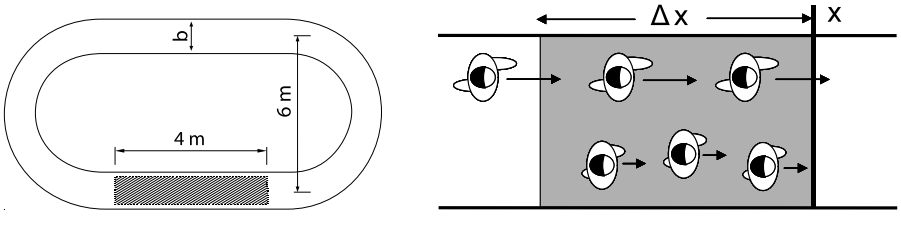


Fig. 1. Left: Experimental setup to determine the fundamental diagram for the movement of pedestrians along a line ($b = 0.7$m). The measurement area is dashed. **Right:** Illustration of different measurement methods to determine the fundamental diagram. Local measurements at cross-section with position x averaged over a time interval Δt have to be distinguished from measurements at certain time averaged over space Δx.

Method A is based on local measurements of the observable O at a certain location x, averaging then over a time interval Δt. Such averages will be denote

by $\langle O\rangle_{\Delta t}$. Measurements at a certain location allow a direct determination of the flow J and the velocity v:

$$\langle J\rangle_{\Delta t} = \frac{N}{\Delta t} = \frac{1}{\langle \Delta t_i\rangle_{\Delta t}} \qquad \text{and} \qquad \langle v\rangle_{\Delta t} = \frac{1}{N}\sum_{i=1}^{N} v_i\,. \tag{5}$$

The flow is given as the number of persons N passing a specified cross-section at x per unit time. Usually it is taken as a scalar quantity since only the flow normal to the cross-section is considered. To relate the flow with a velocity one measures the individual velocities v_i at location x and calculates the mean value of the velocity $\langle v\rangle_{\Delta t}$ of the N pedestrians. In principle it is possible to determine the velocities v_i and crossing times t_i of each pedestrian and to calculate the time gaps $\Delta t_i = t_{i+1} - t_i$ defining the flow as the inverse of the mean value of time gaps over the time interval Δt.

Method B averages the observable O over space Δx at a specific time t_k which gives $\langle O\rangle_{\Delta x}$. By introducing an observation area with extend b, Δx the density ρ and the velocity v can be determined directly:

$$\langle \rho\rangle_{\Delta x} = \frac{N'}{b\,\Delta x} \qquad \text{and} \qquad \langle v\rangle_{\Delta x} = \frac{1}{N'}\sum_{i=1}^{N'} v_i\,. \tag{6}$$

This method was used in combination with time-lapse photos. Due to cost reasons often only the velocity of single pedestrians and the mean value of the velocity during the entrance and exit times were considered [6,7].

Flow equation: The hydrodynamic equation $J = \rho\, v\, b$ allows to relate these methods and to change between different representations of the fundamental diagram. It is possible to derive the flow equation from the definition of the observables introduced above by using the distance $\Delta\tilde{x} = \Delta t\,\langle v\rangle_{\Delta t}$. Thus one obtains

$$J = \frac{N}{\Delta t} = \frac{N}{b\Delta\tilde{x}}\frac{b\Delta\tilde{x}}{\Delta t} = \tilde{\rho}\; b\; \langle v\rangle_x \qquad \text{with} \qquad \tilde{\rho} = \frac{N}{b\Delta\tilde{x}}. \tag{7}$$

At this point it is crucial to note that the mean values $\langle v\rangle_x$ and $\langle v\rangle_t$ do not necessarily correspond. This is illustrated by Fig. 1. The upper lane consists of faster pedestrians than the lower lane. Averaging over Δx does not consider the last pedestrian in the lane above while avereraging over Δt at x_0 for appropriate Δt does. Thus densities calculated by $\tilde{\rho} = \langle J\rangle_{\Delta t}/\langle v\rangle_{\Delta t}$ can differ from direct measurements via $\langle \rho\rangle_{\Delta x}$. We come back to this point later.

As already mentioned above we choose the most simple system for our own experiments [12] to get an estimation for the lower bound of deviations resulting between different measurement methods. To measure the fundamental diagram of the movement along a line we performed 12 runs with varying number of pedestrians, $N = 17$ to $N = 70$. For the movement along a line we set $b = 1$ in the equations introduced above. We note again that the different measurements shown in the next figures are based on the same set of trajectories determined

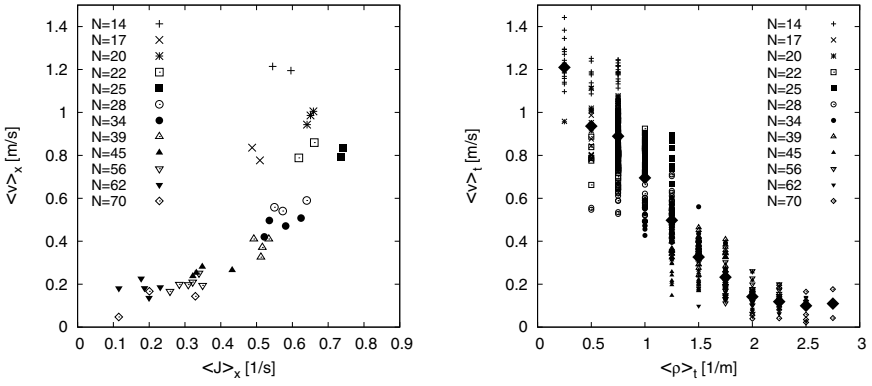


Fig. 2. Fundamental diagrams measured at the same set of trajectories but with different methods. **Left:** Measurement at a certain cross-section averaging over time interval (Method A). **Right:** Measurement at a certain point in time averaging over space (Method B). Large diamonds give the overall mean value of the velocity for one density value.

automatically from video recordings of the measurement area with high accuracy ($x_{\mathrm{err}} \pm 0.02$ m) [11]. The data analysis is restricted to the stationary state.

Fig. 2 shows the direct measurements according to Method A and B. For Method A we choose the position of the cross-section in the middle of the scene $x = 0$ m and a time interval of $\Delta t = 30$s. For Method B the area ranges from $x = -2$ m to $x = 2$ m, and we performed the averaging over space each time t_k a pedestrian crossed $x = 0$. For Method B we note that the fixed length of the observation area of 4 m results in discrete density values with distance $\Delta\rho = (4\mathrm{m})^{-1}$. For each density value large fluctuations of the velocities $\langle v\rangle_t$ are observed. The large diamonds in Fig. 2(right) represent the mean values over all velocities $\langle v\rangle_t$ for one density. The flow equation (7) allows to switch the direct measurement of Method A and B into the most common representation of the fundamental diagram $J(\rho)$.

Fig. 3 shows a comparison of fundamental diagrams using the same set of trajectories but different measurement methods. In particular for high densities, where jam waves are present, the deviations are obvious. For the high density regime the trajectories show inhomogeneities in time and space, which do not correspond, see [14]. The averaging over different degrees of freedom, the time Δt for Method A and the space Δx for Method B leads to different distributions of individual velocities. Thus one reason for the deviations is that the mean values of the velocity measured at a certain location by averaging over time do not necessarily conform to mean values measured at a certain time averaged over space. However, the straightforward use of the flow equation neglects these differences. In [15] it was stated that the difference can be cancelled out by using the harmonic average for the calculation of the mean velocity for Method A. We have tested this approach and found that the differences do not cancel out in general and the data are only in conformance if one takes into account

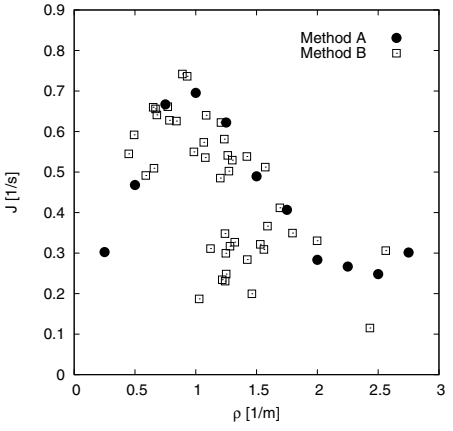


Fig. 3. Fundamental diagram determined by different measurement methods. *Method A*: Direct measurement of the flow and velocity at a cross-section. The density is calculated via $\rho = \langle J \rangle_{\Delta t} / \langle v \rangle_{\Delta t}$. *Method B*: Measurement of the density and velocity at a certain time point averaged over space. The flow is given by $J = \rho \langle v \rangle_{\Delta x}$.

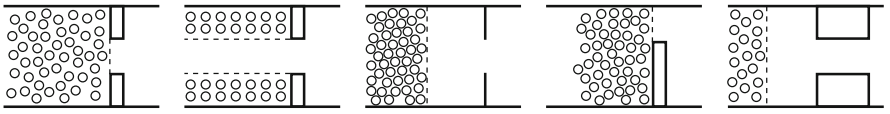
the fluctuations and calculates the mean velocity by the harmonic average. But for states where congestions lead to an intermittent stopping, fluctuations of the density measured with Method A are extremly large and can span over the whole density range observed. This due to the fact that in Method A the density is determined indirectly by calculating $\tilde{\rho} = \langle J \rangle_{\Delta t} / \langle v \rangle_{\Delta t}$. In the high density range the flow as well as the velocity have small values causing high fluctuations for the calculated density.

4 Experiments at Bottlenecks

We now discuss another important scenario, namely the flow of pedestrians through bottlenecks, i.e. areas where the capacity is reduced. It shows a rich variety of phenomena, e.g. the formation of lanes at the entrance to the bottleneck [17,18,19], or clogging and blockages at narrow bottlenecks [2,20,21,22]. Moreover, bottleneck capacities provide important information for the design and dimensioning of pedestrian facilities. In the following we present results of an experiment performed to analyze the influence of the width and the length of a bottleneck.

Fig. 4 shows the different experimental setups used in various studies, as discussed in [19]. It shows considerable differences in the geometry of the bottleneck and the initial distribution of pedestrians. We consider here only the experiments by Kretz [18], Müller [21] and Seyfried [19] (Fig. 4(a), 4(c), 4(e)) which study the motion of an evenly distributed group through a symmetrical bottleneck.

Our experiment was performed in 2006 in the wardroom of the "Bergische Kaserne Düsseldorf" with a test group that was comprised of soldiers. The

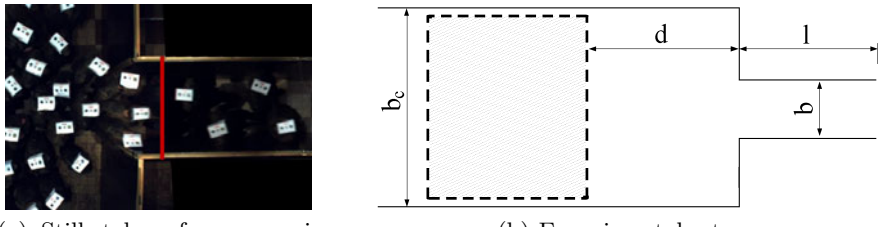


(a) Kretz [18]. (b) Muir [22]. (c) Müller [21]. (d) Nagai [23]. (e) Seyfried [19].

Fig. 4. Experimental setups used in various studies of bottleneck flow

experimental setup allows to probe the influence of the bottleneck width and length. In one experiment the width b was varied (from 90 to 250 cm) at fixed corridor length, the other experiment investigated varying the corridor length l $(6, 200, 400$ cm) at fixed $b = 160$ cm. Wider bottlenecks with more test persons were studied than in previous experiments.

Fig. 5(b) shows a sketch of the experimental setup used to analyze the flow through bottlenecks and Fig. 5(a) a still taken from the experiment. To ensure an equal initial density for every run, holding areas were marked on the floor (dashed regions). All together 99 runs with up to 250 people were performed over five days.



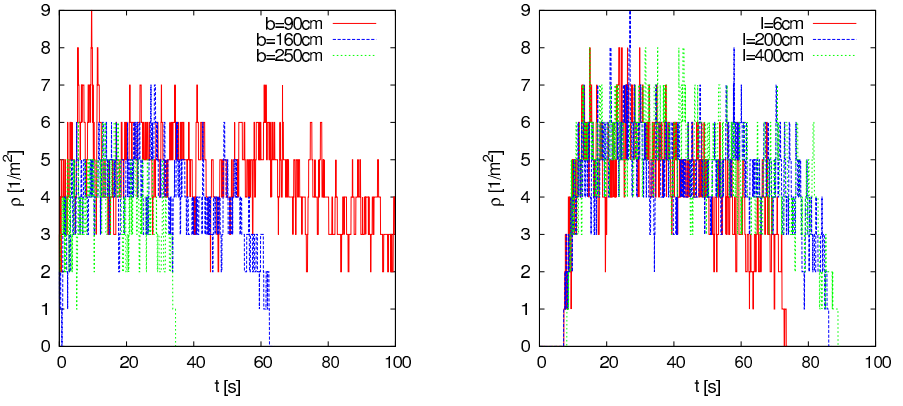
(a) Still taken from experiment. (b) Experimental setup.

Fig. 5. Experimental setup used in our bottleneck experiments

Fig. 6 shows the time evolution of the density in front of the bottleneck. In the bottleneck width experiment the dependence on b is immediately obvious (Fig. 6(a)) with the highest densities being seen in front of the narrowest bottlenecks (between 20 and 30 seconds the densities are 5.46 ± 0.75, 4.79 ± 1.00 and $3.52 \pm 0.87\,[m^{-2}]$ for $b = 90, 160, 250$ cm, respectively. For experiments with different bottleneck lengths no appreciable difference in the densities can be observed, Fig. 6.

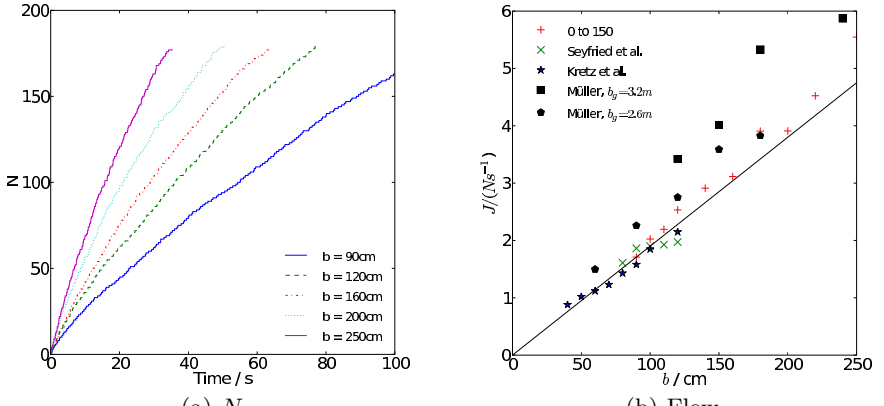
Due to the accurate trajectories that we could determine using an automated video analysis, the process of lane formation can be studied in detail.

In Figs. 7(a) and 8(a), N the total number of pedestrians passing the measurement line can be seen to curve downwards. Therefore the flow is time-dependent, diminishing as the experiment runs its course. The flow will depend on the number of pedestrians considered. In this analysis we calculate the flow using the first 150 people. Previous experiments have not observed this time dependence,



(a) Dependence on b. (b) Dependence on l.

Fig. 6. Time evolution of the density in front of bottleneck

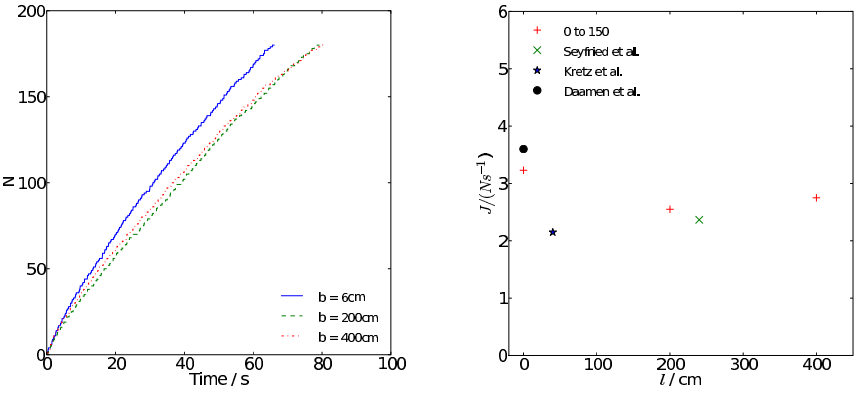


(a) N. (b) Flow.

Fig. 7. (a) Total number N of pedestrians passing the measurement line and (b) variation of the flow J with bottleneck width b

as the participation in the experiments was not high enough (for example in [24] only 100 pedestrians took part). As expected the flow exhibits a strong dependence on the width of the bottleneck b, see Fig. 7. The bottleneck length l exerts virtually no influence on the flow, except for the case of an extremely short constriction (Fig. 7) where three lanes can be formed. In Figs. 7(b) and 8(b), the flow from our experiment is compared with previous measurements. The black line in Fig. 7(b) represents a constant specific flow of 1.9 $(\mathrm{ms})^{-1}$. The difference between the flow at $l = 6$ cm and $l = 200, 400$ cm is $\Delta J \simeq 0.5\ \mathrm{s}^{-1}$.

The datapoints of Müller's experiments lie significantly above the black line. The Müller experimental setup features a large initial density of around 6 Pm^{-2} and an extremely short corridor. The discrepancy between the Müller data and



(a) N. (b) Flow.

Fig. 8. (a) Total number N of pedestrians passing the measurement line and (b) variation of the flow J with bottleneck length l

the empirical $J = 1.9b$ line is roughly $\Delta J \simeq 0.5$ s^{-1}. This difference can be accounted for due to the short corridor, but may also be due to the higher initial density in the Müller experiment.

5 Conclusions

In this contribution we have discussed the fact that our understanding of pedestrian dynamics suffers from the lack of consensus about empirical results. This concerns even basic qualitative properties of the most relevant quantities, like the fundamental diagram or bottleneck flows.

One factor are the different measurement methods used in various experimental studies. We have argued that this can have a considerable influence on the results and makes the comparison of different experiments difficult.

One promising method to obtain reliable and reproducable empirical data are large scale experiments under controlled conditions. We have presented results from a recent study, focussing on the bottleneck scenario. Using a technique to determine the positions of the pedestrians automatically provided us with high quality data for the trajectories. This allows reliable conclusions about the behaviour. As an example, strong evidence for a continuous, in contrast to a stepwise, increase of the bottleneck capacity with the width has been found.

References

1. Schadschneider, A., et al.: Evacuation dynamics: Empirical results, modeling and applications. In: Meyers, R.A. (ed.) Encyclopedia of Complexity and System Science. Springer, Heidelberg (2008)
2. Predtechenskii, V.M., Milinskii, A.I.: Planing for foot traffic flow in buildings. Amerind Publishing, New Dehli (1978)

3. Weidmann, U.: Transporttechnik der Fussgänger. Schriftenreihe des IVT Nr. 90, ETH Zürich (1993)
4. Nelson, H.E., Mowrer, F.W.: Emergency movement. In: DiNenno, P.J. (ed.) SFPE Handbook of Fire Protection Engineering, 3rd edn. (2002)
5. Hankin, B.D., Wright, R.A.: Passenger Flow in Subways. Operational Research Quarterly 9, 81–88 (1958)
6. Older, S.J.: Movement of Pedestrians on Footways in Shopping Streets. Traffic Engineering and Control 10, 160–163 (1968)
7. Navin, P.D., Wheeler, R.J.: Pedestrian flow characteristics. Traffic Engineering 39, 31–36 (1969)
8. Mori, M., Tsukaguchi, H.: A new method for evaluation of level of service in pedestrian facilities. Transp. Res. 21A(3), 223–234 (1987)
9. Helbing, D., et al.: Dynamics of Crowd Disasters: An Empirical Study. Phys. Rev. E 75, 046109 (2007)
10. Seyfried, A., Boltes, M., Kähler, J., Klingsch, W., Portz, A., Rupprecht, T., Schadschneider, A., Steffen, B., Winkens, A.: Enhanced empirical data for the fundamental diagram and the flow through bottlenecks. In: Pedestrian and Evacuation Dynamics 2008, p. 133. Springer, Heidelberg (2010) (in Print)
11. Boltes, M., Seyfried, A., Steffen, B., Schadschneider, A.: Automatic Extraction of Pedestrian Trajectories from Video Recordings. In: Pedestrian and Evacuation Dynamics 2008, p. 39. Springer, Heidelberg (2010) (in Print)
12. http://www.fz-juelich.de/jsc/math/RD/projects/ped_dynamics/
13. Fruin, J.J.: Pedestrian Planning and Design. In: Metropolitan Association of Urban Designers and Environmental Planners, New York (1971)
14. Portz, A., Seyfried, A.: Modeling Stop-and-Go Waves in Pedestrian Dynamics. In: Wyrzykowski, R., et al. (eds.) PPAM 2009, Part II. LNCS, vol. 6068, pp. 561–568. Springer, Heidelberg (2010)
15. Leutzbach, W.: Introduction to the Theory of Traffic Flow. Springer, Berlin (1988)
16. Kerner, B.S.: The Physics of Traffic. Springer, Berlin (2004)
17. Hoogendoorn, S., Daamen, W.: Pedestrian Behavior at Bottlenecks. Transp. Sc. 39(2), 147–159 (2005)
18. Kretz, T., Grünebohm, A., Schreckenberg, M.: Experimental study of pedestrian flow through a bottleneck. J. Stat. Mech. 10, P10014 (2006)
19. Seyfried, A., Passon, O., Steffen, B., Boltes, M., Rupprecht, T., Klingsch, W.: New insights into pedestrian flow through bottlenecks. Transp. Sc. 43, 395–406 (2009)
20. Dieckmann, D.: Die Feuersicherheit in Theatern, Jung, München (1911) (in German)
21. Müller, K.: Die Gestaltung und Bemessung von Fluchtwegen für die Evakuierung von Personen aus Gebäuden. Dissertation, Technische Hochschule Magdeburg, Vorlage (1981)
22. Muir, H., Bottomley, D., Marrison, C.: Effects of Motivation and Cabin Configuration on Emergency Aircraft Evacuation Behavior and Rates of Egress. Int. Jour. Aviation Psychology 6, 57–77 (1996)
23. Nagai, R., Fukamachi, M., Nagatani, T.: Evacuation of crawlers and walkers from corridor through an exit. Physica A 367, 449–460 (2006)
24. Daamen, W., Hoogendoorn, S.: Capacity of doors during evacuation conditions. In: Proc. 1st Int. Conf. on Evacuation Modeling and Management (2009)

Properties of Safe Cellular Automata-Based S-Boxes

Miroslaw Szaban[1] and Franciszek Seredynski[2,3]

[1] Institute of Computer Science, University of Podlasie, 3-go Maja 54, Siedlce, Poland
mszaban@ap.siedlce.pl
[2] Institute of Computer Science, Polish Academy of Sciences,
Ordona 21, 01-237 Warsaw, Poland
[3] Polish-Japanese Institute of Information Technology
Koszykowa 86, 02-008 Warsaw, Poland
sered@ipipan.waw.pl

Abstract. In the paper we use recently proposed cellular automata (CA) - based methodology [9] to design $8 \times n$ ($n \leq 8$) S-boxes functionally equivalent to S-boxes used in current cryptographic standards. We provide an exhaustive experimental analysis of the proposed CA-based S-boxes in terms of non-linearity, autocorrelation and scalability, and compare results with other proposals. We show that the proposed CA-based S-boxes have cryptographic properties comparable or better than currently offered classical S-box tables.

Keywords: Cellular Automata, S-boxes, Block Cipher, Cryptography.

1 Introduction

In the modern world, two main cryptography approaches are used today to provide a secure communication: secret key and public key systems. The main concern of this paper are cryptosystems with a secret key. The main interest of this work are CA and their application to design S-boxes. S-boxes functionally realize some Boolean functions, important from point of view of requested cryptographic features in secret key systems.

Many known secure standards of symmetric key cryptography, such as, e.g. presented in [3], [4], use efficient and secure algorithms working on the base of S-boxes. S-boxes are ones of the most important components of block ciphers, which are permanently upgraded, or substituted by new better constructions.

In the next section the concept of the S-box and its most known applications in DES cryptographic standards are presented. Section 3 describes the main cryptographic criteria to examine Boolean functions. In section 4 two different Boolean functions describing the work of S-boxes are proposed and measures of evaluating their cryptographical properties such as non-linearity and autocorrelation. Section 5 outlines the concept of CA and the idea of creating CA-based S-boxes. Section 6 presents results of examination of cryptographic features of CA-based S-boxes and their comparison with other proposals. The last section concludes the paper.

R. Wyrzykowski et al. (Eds.): PPAM 2009, Part II, LNCS 6068, pp. 585–592, 2010.

2 S-Boxes in Cryptography

S-box (see, [3]) is a function $f : B^n \rightarrow B^k$, which from each of n Boolean input values of B^n block consisting of n bits b_i $(i \leq n)$ generates some k Boolean output values called B^k block consisting of k bits b_j $(j \leq k$ and $k \leq n)$, what corresponds to the mapping bit strings $(b_0, b_1, ..., b_n) \rightarrow (b_0, b_1, ..., b_k)$. When n is equal to k, the function f, from n different input values maps n different outputs values, and such a S-box is called bijective [5].

Let us note that the classical S-boxes (DES and its successor AES, see, [4]) are fixed, not flexible structures requesting predefined sizes of memory. Therefore, it is hard to use them in new designed cryptographic algorithms, which request using dynamic S-boxes. The purpose of this study is to design flexible S-boxes, ready to use in cryptographic algorithms with dynamic S-boxes.

3 Preliminaries for Evaluation of Boolean Functions

The quality of S-boxes designed with use of CA must be verified by required properties of S-boxes. The most important definitions and dependencies related to this issue are recalled below from cryptographic literature [1], [2], [10], [11].

A Boolean function $f : Z_2^n \rightarrow Z_2$ maps n binary inputs to a single binary output. The list of the size of 2^n of all possible outputs is the *truth table*. *Polarity* form of *the truth table* is denoted by $\hat{f}(x)$ and defined as: $\hat{f}(x) = (-1)^{f(x)}$.

The non-linearity N_f of a Boolean function f is the minimal distance of the function f to the set of affine functions and is calculated as:

$$N_f = \frac{1}{2}(2^n - WH_{max}(f)), \tag{1}$$

where the WH_{max} is the maximal value of the Walsh Hadamard Transform. Ciphers with high non-linearity (low WH_{max}) are known to be more difficult to cryptanalysis.

The next important property of ciphers is autocorrelation AC_f. Autocorrelation defines correlation between polar form $f(x)$ and its polar *shifted version*, $f(x \otimes s)$. Autocorrelation of a Boolean function f is defined by Autocorrelation Transform given by the equation: $\hat{r}_f(s) = \sum_x \hat{f}(x)\hat{f}(x \otimes s)$, where $s \in Z_2^n - \{0\}$. The absolute maximum value of any autocorrelations is denoted by the equation:

$$AC_f = max_{s \neq 0} |\sum_x \hat{f}(x)\hat{f}(x \otimes s)|. \tag{2}$$

Ciphers with low autocorrelation are known to be more secure.

4 Measuring Cryptographic Properties of S-Boxes

S-boxes as functions mapping n input bits into k output bits under condition $k \leq n$, generally do not satisfy conditions to be a Boolean function, because the

number of output bits of S-boxes is usually higher then one bit ($1 \leq k$). However, the quality of block ciphers received with use of S-boxes is usually measured by criteria proper to Boolean functions. The question which arises is how to apply these criteria to S-boxes (block ciphers). Let us consider two possible methods to solve this problem.

4.1 Method 1: Linear Combination of Single-Output S-Boxes

A Boolean function returns as output one bit. To use Boolean functions criteria to examine S-boxes, we need to transform all k bits output of an S-box into one output bit. After this modification, we obtain a new Boolean function which can be defined as: $f_\beta : B^n \rightarrow B^1$, and expressed by the formula (see, [1], [8]):

$$f_\beta(x) = \beta_1 f_1(x) \otimes \beta_2 f_2(x) \otimes ... \otimes \beta_k f_k(x). \tag{3}$$

The new function is a linear combination of k functions $f_i(x)$, $i \leq k$, where $\beta_i \in B^k$. Each of these functions is defined as a simple S-box (single-output S-box, a part of the $n \times k$ S-box). The relationship (vector $(\beta_1, ..., \beta_k)$) between simple S-boxes is used as a result of the S-box table composition.

Under this approach cryptographical properties of S-boxes presented in section 3 are calculated with use of the Boolean function $f_\beta(x)$.

4.2 Method 2: Set of Single-Output S-Boxes

In this method the S-box is considered as a set of simple S-boxes. A simple S-box satisfies conditions to be a Boolean function. Each simple S-box is a function: $f_i : B^n \rightarrow B^1$, where $i = 1, 2, ..., k$ and the 1-bit output is one of k output bits of the S-box.

Cryptographic properties of an S-box are measured under this method separately for each simple S-box ($\{f_1, ..., f_k\}$), where k is the number of output bits in the $n \times k$ S-box. To analyze the $n \times k$ S-box, using the single-output S-box method, the k single-output S-boxes are considered. It results in analyzing the k Boolean functions corresponding to single-output S-boxes and evaluating their non-linearities and autocorrelations. Partial results are compared and the worst ones become the final evaluation of the analyzed S-box. Such an approach was recently used in cryptanalysis of known ciphers or in [7] to analyze the 6×6 S-boxes.

5 CA-Based S-Boxes Designing

5.1 The Concept of Cellular Automata

One dimensional (1D) CA is in the simplest case a collection of two-state elementary cells arranged in a lattice of the length N, and locally interacting in a discrete time t. For each cell i called a central cell, a neighbourhood of a radius r is defined, consisting of $n_i = 2r+1$ cells, including the cell i. When considering a

finite size of CA, and a cyclic boundary condition is applied, it results in a circle grid. It is assumed that a state q_i^{t+1} of a cell i at the time $t+1$ depends only on states of its neighbourhood at the time t, i.e. $q_i^{t+1} = TF(q_i^t, q_{i1}^t, q_{i2}^t, , q_{in}^t)$, and a transition function TF, called a rule, which defines a rule of updating a cell i. A length L of a rule and a number of neighbourhood states for a binary uniform CA is $L = 2^n$, where $n = n_i$ is a number of cells of a given neighbourhood, and a number of such rules can be expressed as 2^L.

5.2 Construction of CA-Based S-Boxes

In our study we propose to use CA as a function which can be characterized by the same properties and realize the same functions as wide known S-boxes. A motivation for applying CA to realize S-boxes steams from potentially very interesting features of CA. CA have a computational possibilities equivalent to Universal Turing Machine, what means that such Boolean functions can be realized.

A classic S-box is a function expressed as a table containing natural numbers. Cryptographic literature shows many examples and methods of searching S-box tables. The quality of S-boxes are measured with use of different functions which examine their different properties [6], [1], [8], [10], [11]. Some of the most important test functions were presented in section 3. In [6], [1], [8], [5], authors treat the problem of designing S-box tables as a combinatorial optimization problem and apply different metaheuristics to search solutions in the huge space of S-box tables solutions. Recently [9] we have proposed CA-based approach to create S-boxes in the form not tables, but some virtual entities.

The CA-based S-box can be seen as CA composed of the following elements:

- a number of CA cells performing the role of background
- a number of CA cells performing the role of input/output of CA-based S-box
- an initial state of CA
- an appropriate rule/rules of CA.

It is assumed that CA will evolve during a number of time steps. Selected cells of CA (in its initial state) serve as input bits of the S-box, and the same cells, after declared time steps, are considered as the output of the S-box. To construct CA performing the S-box function it is necessary to find appropriate CA rules and verify produced results according to the S-box functions criteria.

The first step in constructing the $n \times k$ CA-based S-box is selecting a number of CA cells. A number of CA cells must be not lower than $max|n, k|$. This number should be also enough large to generate cycles longer than the number of CA time steps [7]. Construction of $n \times n$ S-box is simple (see, [9]), because we can use n cells of CA and from n inputs we obtain n outputs. How to use CA to construct $n \times k$ S-boxes, when $n \geq k$? For this purpose we propose to consider the first n CA cells at the time step $t = 0$ as input cells, and the first k CA cells at the last time step as output cells.

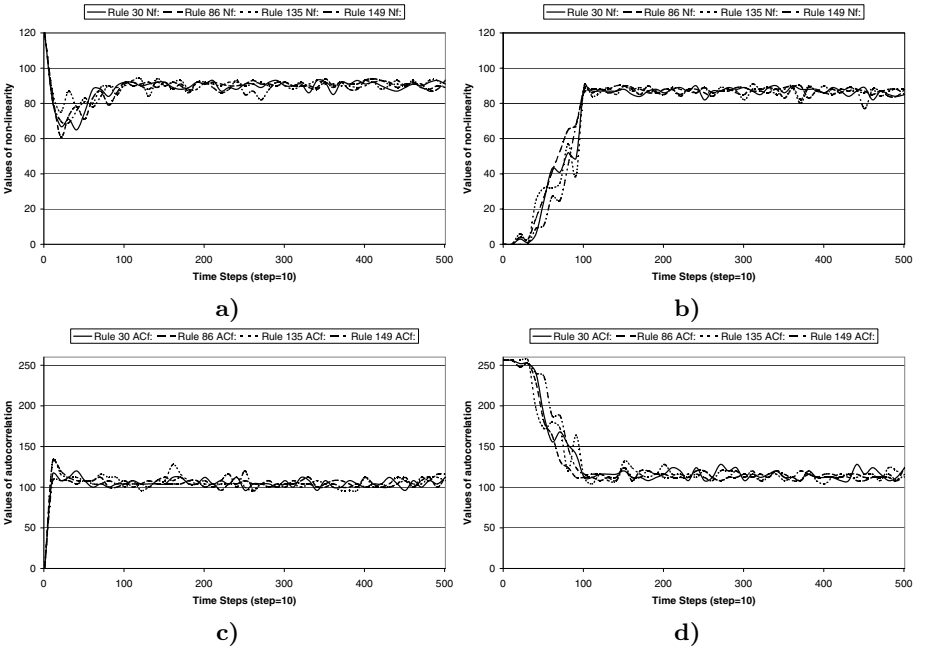


Fig. 1. Evaluation cryptographic properties of 8×8 CA-based S-boxes (the worst from 1000 random initial states) in time steps for selected CA rules. Non-linearity: Method 1 (a) and Method 2 (b); autocorrelation: Method 1 (c) and Method 2 (d).

In [9] we used quite long CA (with a number CA cells = 100), where significant input/output-bit cells collectively cooperate with other cells called the background. This construction is large enough to satisfy condition for non-cycle construction. The number of time steps which satisfy this condition is equal to 100 (see, [9]). The number of time steps was also determined by behavior of CA in different time steps. Experiments (see, Fig. 1) show that CA provides stable and highest quality of cryptographic properties (Fig. 1a - non-linearity, Fig. 1c - autocorrelation, calculated by Method 1, and Fig. 1b - non-linearity, Fig. 1d - autocorrelation, calculated by Method 2) for time steps not lower than 100. In the next study the number of time steps will be equal to 100.

Not every CA rule is suitable to provide proper quality for CA-based S-box. From the all set of 256 elementary CA rules (CA rules with neighborhood radius $r = 1$), we selected four rules $\{30, 86, 135, 149\}$ as only proper for this purpose (see also, [9]).

The initial state of CA (the first n bits interpreted as the S-box input) is randomly set and CA starts to run. After a predefined number of time steps the CA stops and after 100 time steps the first k bits are treated as output bits, which are next used to evaluate quality of CA-based S-boxes.

6 Analysis and Comparison of S-Boxes

6.1 $8 \times k$ S-Boxes Analysis

In [9] we proposed the 8×8 CA-based S-boxes, which offer cryptographic quality in general comparable or better then nowadays constructed S-box tables. To complement our proposition it is needed to study $8 \times k$ (where $k \leq 8$) CA-based S-boxes and compare results with other approaches. We analyze CA-based S-boxes composed of: 100 CA cells (initial state), 100 time steps and CA rules 30, 86, 135 and 149 (only these rules are proper for this purpose). During experiments we analyzed 10000 randomly selected CA-based S-boxes (initial states of CA) and calculated their non-linearities and autocorrelations, for each of CA rules.

Table 1. The best values of non-linearity and autocorrelation (N_f, AC_f) of $8 \times k$ CA-based S-boxes and other proposals, calculated by Method 1. * - no data.

Input × Output	Millan et al. [6]	Clark et al. [1]	Nedjah, Mourelle [8]	Our approach: CA-based S-box Rule 30, Rule 86, Rule 135, Rule 149
8×2	(110, 48)	(114, 32)	(116, 34)	(111, 44), (110, 44), (111, 44), (111, 44)
8×3	(108, 56)	(112, 40)	(114, 42)	(111, 44), (110, 44), (110, 44), (110, 44)
8×4	(106, 64)	(110, 48)	(110, 42)	(111, 44), (111, 40), (110, 44), (110, 44)
8×5	(104, 72)	(108, 56)	(110, 56)	(111, 44), (111, 44), (111, 44), (111, 44)
8×6	(104, 80)	(106, 64)	(106, 62)	(110, 40), (110, 40), (110, 40), (110, 40)
8×7	(102, 80)	(104, 72)	(102, 70)	(110, 44), (111, 40), (112, 40), (112, 40)
8×8	(100, *)	(102, 80)	(*, *)	(111, 44), (111, 44), (111, 44), (111, 44)

Table 1 shows the best values of non-linearity and autocorrelation for S-box tables discovered by Millan et al. [6], Clark et al. [1], Nedjah and Mourelle [8], and our CA-based S-boxes (with four CA rules). Presented results were obtained with use of the Method 1, which constructs a Boolean function from single S-boxes (see, section 4). Heuristic methods of search the best S-box tables such as Genetic Algorithm or Simulated Annealing were used by these authors. Results were calculated with use of all possible linear combination of simple s-boxes.

One can see that our results presented in Table 1 are always better than Millan et al. results (Millan best result is N_f=110, AC_f=48, but our result is N_f=110, AC_f=44 for $k = 2$). Clark et al. and Nedjah & Mourelle obtained a bit better values than our proposal for output bits $k < 4$. In other cases ($k \geq 4$) our approach gives higher N_f and lower AC_f. Advantage of CA-based S-boxes over other approaches grows with growing the number of output bits.

Another observation is that our approach gives stable values of non-linearity (ranging {110, 112}) and autocorrelation (ranging {40, 44}) independently on the number of outputs (and used CA rules) in opposite to another approaches where calculated values became worse with growing number of outputs.

Table 2 presents values of non-linearity and autocorrelation for CA-based S-boxes calculated by two different methods (see, section 4). Values of non-linearity

Table 2. The best values of non-linearity and autocorrelation (N_f, AC_f) of $8 \times k$ CA-based S-boxes and other proposals, calculated by two methods

In. × Out.	Method 1 - lin. comb. of simple s-boxes Rule 30, Rule 86, Rule 135, Rule 149	Method 2 - simple s-boxes Rule 30, Rule 86, Rule 135, Rule 149
8×2	(111, 44), (110, 44), (111, 44), (111, 44)	(108, 48), (109, 48), (108, 48), (109, 48)
8×3	(111, 44), (110, 44), (110, 44), (110, 44)	(107, 56), (108, 56), (108, 56), (108, 52)
8×4	(111, 44), (111, 40), (110, 44), (110, 44)	(107, 56), (107, 56), (107, 56), (107, 56)
8×5	(111, 44), (111, 44), (111, 44), (111, 44)	(106, 60), (107, 56), (106, 56), (107, 56)
8×6	(110, 40), (110, 40), (110, 40), (110, 40)	(106, 60), (106, 60), (106, 56), (106, 56)
8×7	(110, 44), (111, 40), (112, 40), (112, 40)	(106, 60), (105, 64), (105, 60), (105, 60)
8×8	(111, 44), (111, 44), (111, 44), (111, 44)	(105, 64), (105, 64), (105, 60), (105, 60)

in Method 2 is lower than in Method 1, and values of autocorrelation in Method 2 is higher than in Method 1, because in Method 2 we analyze simple S-boxes and the worst values expressed quality of examined S-box.

6.2 Scalability of $n \times n$ S-Boxes

In this section we study the scalability of $n \times n$ S-boxes. Table 3 shows values of nonlinearity and autocorrelation for different sizes of input/output bits of S-box.

Table 3. The best values of non-linearity and autocorrelation (N_f, AC_f) of $n \times n$ CA-based S-boxes (calculated for 1000 CA) and other proposals, calculated by linear combination of simple s-boxes. * - no data.

Input × Output	Millan [5]	Clark et al. [1]	Our approach: CA-based S-box Rule 30, Rule 86, Rule 135, Rule 149
5×5	(10, *)	(10, 16)	(12, 8), (12, 8), (12, 8), (12, 8)
6×6	(20, *)	(22, 32)	(26, 16), (26, 12), (25, 12), (25, 16)
7×7	(46, *)	(48, 48)	(53, 24), (53, 28), (53, 24), (53, 24)
8×8	(100, *)	(102, 80)	(111, 44), (111, 44), (111, 44), (111, 44)
10×10	(*, *)	(*, *)	(470, 112), (470, 112), (469, 116), (470, 112)
12×12	(*, *)	(*, *)	(1946, 280), (1945, 284), (1946, 280), (1948, 284)

One can see in Table 3 that, CA-based S-box keeps higher non-linearity and lower autocorrelation than Millan and Clark et al. results, despite of the number of inputs/outputs of S-box. The third column in Table 3 presents values of non-linearity and autocorrelation for $n \times n$ CA-based S-boxes. One can see that with the growth of n, values for non-linearity also grow proportionally and values of autocorrelation, proportionally fall down. It means that CA-based S-boxes characterize by proportionally better values of non-linearity and autocorrelation for higher dimensions of S-box (values are better with growing n).

7 Conclusions

The paper presents an idea of creating S-boxes using CA-based approach. Classical S-boxes based on tables are fixed structure constructions. We are interested in creating CA-based S-boxes, which are dynamical structures. CA from input block of bits generates output block of bits and is evaluated by the same examine criteria like the traditional S-box. Conducted experiments have shown that the $8 \times k$ CA-based S-boxes are characterized by a high non-linearity and low autocorrelation independently on the method of calculation. These values in many cases are better than classical tables of S-boxes. CA-based S-boxes keep the property of scalability. It means that for higher dimensions of S-box it is characterized by proportionally better values of nonlinearity and autocorrelation. They are very well suited for cryptographic systems with dynamic S-boxes.

Acknowledgements

This work was supported by Polish Ministry of Science and Higher Education as the grant No. N N519 388036.

References

1. Clark, J.A., Jacob, J.L., Stepney, S.: The Design of S-Boxes by Simulated Annealing. New Generation Computing 23(3), 219–231 (2005)
2. Dowson, E., Millan, W., Simpson, L.: Designing Boolean Functions for Cryptographic Applications. Contributions to General Algebra 12, 1–22 (2000)
3. Federal Information Processing Standards Publication, Fips Pub 46-3, DES (1999), http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf
4. Federal Information Processing Standards Publications, FIPS PUBS 197, AES (2001), http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
5. Millan, W.: How to Improve the Non-linearity of Bijective S-boxes. LNCS, vol. 143, pp. 181–192. Springer, Heidelberg (1998)
6. Millan, W., Burnett, L., Carter, G., Clark, A., Dawson, E.: Evolutionary Heuristics for Finding Cryptographically Strong S-Boxes. In: Varadharajan, V., Mu, Y. (eds.) ICICS 1999. LNCS, vol. 1726, pp. 263–274. Springer, Heidelberg (1999)
7. Mukhopadhyay, D., Chowdhury, D.R., Rebeiro, C.: Theory of Composing Nonlinear Machines with Predictable Cyclic Structures. In: Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 210–219. Springer, Heidelberg (2008)
8. Nedjah, N., de Macedo Mourelle, L.: Designing Substitution Boxes for Secure Ciphers. International Journal Innovative Computing and Application 1(1), 86–91 (2007)
9. Szaban, M., Seredynski, F.: Cryptographically Strong S-Boxes Based on Cellular Automata. In: Umeo, H., Morishita, S., Nishinari, K., Komatsuzaki, T., Bandini, S. (eds.) ACRI 2008. LNCS, vol. 5191, pp. 478–485. Springer, Heidelberg (2008)
10. Webster, A.F., Tavares, S.: On the Design of S-Boxes. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 523–534. Springer, Heidelberg (1986)
11. Youssef, A., Tavares, S.: Resistance of Balanced S-boxes to Linear and Differential Cryptanalysis. Information Processing Letters 56, 249–252 (1995)

Author Index